Banco de tiempo Peer-to-Peer

Antonio Núñez Guerrero
Daniel Alejandro Nowendsztern
Marcos Pérez García

GRADO EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

TRABAJO DE FIN DE GRADO EN INGENIERÍA INFORMÁTICA

Madrid, 20 de junio de 2015
Director: Simon Pickin

Dedicado a
nuestras familias
y personas queridas

Agradecimientos

Al término de esta etapa de nuestras vidas, queremos expresar un profundo agradecimiento a quienes con su ayuda, apoyo y compresión nos alentaron a lograr este objetivo. Concretamente, agradecemos a nuestros familiares y amigos por el cariño y apoyo moral a lo largo de todos estos años de carrera.

Con especial mención a nuestro director de proyecto, el Dr. Simon Pickin que nos ha guiado en el desarrollo de este proyecto.

Resumen

En los últimos quince años se han realizado muchos artículos que prometen un futuro brillante a la tecnología P2P. Sin embargo, este futuro todavía no se ha alcanzado. Después de la primera explosión de interés que tuvo lugar a principios de este siglo, la tecnología P2P ha ido avanzado muy lentamente, y en consecuencia, el campo de las aplicaciones P2P no ha conseguido madurar lo suficiente.

A pesar de los avances en el uso de las tecnologías P2P en sistemas masivos internos de grandes empresas tales como Facebook [43] y LinkedIn [68], y en aplicaciones muy populares como BitTorrent [17] y Bitcoin [15], la tecnología P2P sigue sin estar lo suficientemente preparada como para poder usarse como base para la construcción de una gran variedad de aplicaciones.

En este contexto, el papel habitual de las universidades y centros de investigación es llevar a cabo estudios y construir prototipos que busquen avanzar en los aspectos fundamentales de las tecnologías, con el fin último de poder construir plataformas fiables y seguras sobre las que desarrollar aplicaciones de todo tipo.

Es lógico que en las universidades y centros de investigación no se dedique mucho esfuerzo al estudio de las aplicaciones que podrían beneficiarse de una plataforma P2P fiable y seguro, salvo quizás para definir unos casos de uso demostrativos. Pero este hecho contribuye a alimentar el problema de la gallina y el huevo al que siempre se enfrentan las nuevas tecnologías de base. Por un lado, es difícil justificar la inversión de esfuerzo en el desarrollo de plataformas, si no se pueden exhibir aplicaciones útiles que se ejecutarían sobre ellas. Pero por otro lado, es difícil justificar la inversión de esfuerzo en la construcción de aplicaciones pensadas para ejecutarse sobre plataformas

poco maduras o incluso todavía sin existir.

Una forma de contribuir al avance del campo de la tecnología P2P, consistiría en estudiar las aplicaciones para las que resulta ventajoso ejecutarse sobre una plataforma P2P fiable y segura e intentar especificar de manera precisa los requisitos que imponen estas aplicaciones en la misma. Este proyecto se enmarca dentro de este enfoque.

En este proyecto se estudia la definición de un banco de tiempo P2P (ejemplo sugerido por los propios alumnos) y como parte de este estudio y a modo de demostración de factibilidad, se implementa una parte del sistema especificado sobre una plataforma P2P existente. Dadas las dificultades de abordar este objetivo en un campo tan inmaduro, conceptualmente difícil y sin fuentes documentales definitivas no cabe esperar como resultado del proyecto un producto software elaborado como es el caso de otros proyectos más convencionales.

Abstract

In the last fifteen years there have been many articles that promise a bright future for the P2P technology. However, this future has not yet been reached. After the first explosion of interest that took place at the beginning of this century, the P2P technology has been improve very slowly, and as a result, the field of P2P applications has not been mature enough.

In spite of advances in the use of the P2P technologies in massive internal systems of big businesses such as Facebook [43] and LinkedIn [68], and very popular in applications like BitTorrent [17] and Bitcoin [15], the P2P technology is still not sufficiently prepared to be able to use as the base for the development of a wide variety of applications.

In this context, the usual role of the universities and research centers is to carry out studies and develop prototypes that seek progress in the fundamental aspects of the technologies, with the ultimate aim of being able to build secure and reliable platform on which to develop all kinds applications.

It's logical that universities and research centers don't spend too much effort to study applications that could benefit themselves from a reliable and secured P2P platform, except perhaps to define a few reporting use cases. But this fact contributes to feed the chicken and egg problem that core technologies always have to face. On the one hand, it's difficult to justify the effort investment in the development of platforms, if they cannot exhibit useful applications that would run on them. But on the other hand, it's difficult to justify the effort investment in the development of applications intended to run on platforms not fully ripe or non-existent.

One way to contribute to the progress of P2P technology, would be to

look into the applications for which it's advantageous to run on this platform and try to accurately specifying the requirements imposed in that technology. This project is under this approach.

This project studies the definition of a P2P time bank (example suggested by the students) and as part of this study and to demo mode of feasibility, implements a part of the specified system on a P2P existing platform. Given the difficulties of addressing this objective in a field as immature, conceptually difficult and without definitive documentary sources could not be expected as a result of the project a software product developed as is the case in other more conventional projects.

Palabras clave

- Banco de tiempo
- P2P
- Sistema de reputación P2P
- Sistema de ficheros P2P
- Criptografía
- Verificación y Validación P2P
- FreePastry
- Open Chord

Keywords

- Time bank
- P2P
- P2P Reputation Management
- P2P File System
- Cryptography
- P2P software Verification and Validation
- FreePastry
- Open Chord

Índice general

Índice de figuras

Índice de tablas

Índice de abreviaturas

P2P peer-to-peer

LETS Local Exchange Trading System

DNS Domain Name System

UUCP Unix to Unix CoPy

ROCS Robust Complementary Community Currency System

TTL Time To Live

SKC Symmetric Key Cryptography

PKC Public-Key Cryptography

PFC Proyecto de fin de carrera

API Application Programming Interface

GPL General Public License

DHT Distributed Hash Table

IDE Integrated Development Environment

UML Unified Modeling Language

XP eXtreme Programming

MVC Model View Controller

PaaS Platform as a service

PGP Pretty Good Privacy

TSA Time Stamping Authority

Capítulo 1

Introducción

El objetivo de este capítulo es que el lector comprenda algunos de los conceptos básicos sobre los que se embarca este proyecto. Por esta razón, se seguirá un orden cronológico en la explicación. Empezando por los antecedentes, para entender el origen de las ideas principales. A continuación se explicarán tanto la motivación como los objetivos propuestos para el proyecto, para finalizar con el alcance, es decir, el punto al que sería posible llegar de lograrse los objetivos propuestos.

1.1. Antecedentes

Tanto la idea de banco de tiempo como la tecnología P2P tienen una historia relativamente reciente. Aunque la idea de usar tiempo como moneda es mucho más antigua. En particular, en 1832, el galés Robert Owen estableció una moneda basada en el tiempo en Birmingham, Inglaterra (en unidades de 1, 2, 5, 10, 20, 40 y 80 horas). La idea fue retomada posteriormente por otros activistas del siglo XIX como John Gray, Pierre Proudhon, Karl Marx en Europa, y Josiah Warren en Estados Unidos [141].

Mucho más adelante, en concreto en 1973, nació la idea de banco de tiempo de la mente de una visionaria japonesa llamada Teruko Mizushima, cuyo objetivo era conseguir que grupos de personas intercambiasen labores de forma voluntaria pensando, en particular, en una especie de seguridad social: cuando tienen tiempo y buena salud, los usuarios del banco de tiempo

ofrecen servicios a los demás y cuando no, los consumen de los demás [71]. Pero el primero en llevarla a cabo y hacerse popularmente conocido por tal hecho fue Edgar Cahn en 1980, que dirigió la idea de Teruko hacia el lado económico usando la misma divisa de intercambio laboral que había ideado Robert Owen, es decir, el tiempo [141].

En 1973, la idea de banco de tiempo nació de la mente de una visionaria japonesa Teruko Mizushima, cuyo objetivo era asegurar que los grupos de personas de trabajo voluntario, con el objetivo principal de la construcción de una especie de seguridad social: cuando tienen tiempo y buena salud, usuarios del banco del tiempo ofrecen sus servicios a los demás y cuando no lo hacen, reciben los servicios de otros [71]. En Estados Unidos, la idea se popularizó (y los términos *"Time Bank"* y *"Time Credit"* se hicieron marca registrada) por Edgar Cahn en 1980. Cahn continúo con esta idea y fundó la organización "Timebanks USA" en 1995 [141]. En 1983 el término LETS (Local Exchange Trading Systems) se acuñó para describir la idea de monedas locales para el intercambio de bienes y servicios [136].

En 1983 se originó el término LETS (Local Exchange Trading Systems) para describir divisas locales para el intercambio de servicios y bienes [136]. Desde principios de este siglo los bancos de tiempo se han ido adentrando en el mundo digital. En la actualidad la aplicación web de banco de tiempo más conocida tiene 31.053 miembros en 461 comunidades y se calcula que han servido durante 1.404.574 horas según hOurworld [55].

En el caso de las tecnologías P2P, las primeras comunicaciones entre máquinas para el intercambio de contenido se realizaban bajo las aplicaciones y protocolos UUCP (Unix to Unix CoPy) que permitían replicar la información de un computador Unix a otro por medio de la red, estas se remontan al año 1978 [127]. Como evolución posterior a UUCP nació Usenet. En 1979, este sistema creado por Tom Truscott y Jim Ellis, permitía enviar artículos, organizados de manera jerárquica, entre los nodos de la red mediante un mecanismo de tipo "flooding", es decir, con una estructura de control descentralizado que por tanto se puede considerar una red P2P [142].

En 1983, Paul Mockapetris, diseñó DNS (Domain Name System), un sistema de nomenclatura jerárquica para computadoras, servicios o cualquier recurso conectado a Internet o a una red privada, cuyos servidores intercam-

bian información entre ellos de manera descentralizada, por lo que se puede considerar que los servidores DNS forman una red P2P [129].

Sin embargo, la tecnología P2P se hace popularmente conocida en 1999, año en el que la primera aplicación para el intercambio de ficheros de música (Napster) inicia su andadura en la red [137]. A día de hoy son millones de usuarios los que usan a diario aplicaciones P2P para el intercambio de ficheros, pero no se conoce ninguna aplicación consolidada de banco de tiempo que use la tecnología P2P para su funcionamiento.

1.2. Motivación

La mayor parte de las implementaciones de bancos de tiempo que existen actualmente están muy marcadas por las siguientes problemáticas:

- Altos costes de mantenimiento. Para dar soporte 24/7 (las 24 horas del día, los 7 días de la semana) a una aplicación que trabaja sobre una red centralizada es necesario disponer de un capital para costear el mantenimiento de los servidores y la administración de los usuarios. Puesto que suelen estar operados por organizaciones sin ánimo de lucro que intentan dar un servicio gratis a sus usuarios, los costes son dificilmente asumibles.
- Problemas de actualización y adaptabilidad. Normalmente las soluciones existentes están implementadas en código cerrado y/o bajo licencias de copyright. Esto dificulta que el software existente sea capaz de actualizarse y evolucionar rápidamente para adaptarse a las circunstancias de cada momento. Y además, la mayoría de las soluciones actuales de código abierto estan construídas sobre bases poco sólidas y poco extensibles.
- Dificultades de registro y acceso. En el registro se busca asegurarse de que el usuario que se registra sea una persona real. Generalmente se le exige muchos datos privados que necesitan ser comprobados. Esto pasa por un proceso manual en el que cada persona que se desea registrar tiene que ser verificada. Este proceso es muy lento para grandes cantidades de personas. El acceso al sistema depende del registro, luego no se sabe si al sistema al que se accede es fiable o no.

- Seguridad. Aunque está claro que con una solución centralizada la seguridad es mucho más abordable, muchas soluciones centralizadas actuales no han dedicado mucho esfuerzo a la seguridad y la protección contra el fraude, partiendo de la suposición de que los usuarios son de confianza y se comportan bien.

En este proyecto se pretende sentar las bases que permitan dar una solución efectiva a todas estas problemáticas.

1.3. Objetivos

La idea principal de este proyecto es usar las tecnologías P2P actuales para especificar y diseñar un sistema e implementar un prototipo de aplicación de banco de tiempo.

A continuación se desglosan los objetivos para este proyecto:

- **Estudio y aprendizaje de la tecnología P2P.**
- **Especificación del banco de tiempo P2P.**
  - **Especificación de las funcionalidades del sistema.**
  - **Especificación de diagramas de caso de uso.**
  - **Especificación de diagramas de secuencia.**
  - **Especificación de diagramas de clases.**
  - **Estructura del sistema.**
- **Implementación del prototipo del banco de tiempo P2P.**
- **Simulación del prototipo del banco de tiempo P2P.**

1.4. Plan de trabajo

En un primer momento se dedicaron los primeros meses a definir la idea del proyecto junto a la especificación del mismo. Una vez que se terminó la definición, la idea era buscar una herramienta P2P capaz de simular nuestro

sistema para obtener unos resultados que se pudieran estudiar.

Ante las dificultades para entender los simuladores P2P se realizó un estudio y aprendizaje de la tecnología P2P, por medio de bibliografía, artículos y otros contenidos de la red. Como consecuencia de estas dificultades se recondujo el rumbo del proyecto hacía la implementación de una parte del prototipo.

Durante los meses siguientes se buscaron múltiples herramientas tanto para modelar como para desarrollar una parte del prototipo especificado al inicio. Con el objetivo de desarrollar esta parte del prototipo se siguió de forma flexible una metodología ágil.

Finalmente, durante los últimos meses se terminó de definir la especificación que fue evolucionando a lo largo del proyecto.

1.5. Alcance

Todo el contenido del proyecto estará disponible de forma gratuita, como software libre, a través de Internet. De tal forma, que si algún desarrollador o un grupo de desarolladores se interesasen por crear una solución de banco de tiempo, pudieran continuar desde la base que se ha desarrollado en este proyecto.

Capítulo 2

Introduction

The goal of this chapter is for the reader to understand some of the basic concepts of this project. For this reason, the explanation follows a chronological order, starting with the background, to understand the origin of the main ideas. We explain both the motives and the proposed goals of the project, finishing with the scope (the point at which it would be possible to achieve the goals proposed).

2.1. Background

Both time banking and P2P technology are relatively recent ideas, although the the basic idea of using time as a currency is much older. In particular, in 1832, the Welshman Robert Owen established a time-based currency in Birmingham, England (in units of 1, 2, 5, 10, 20, 40, and 80 hours). The idea was later taken up by other nineteenth century activists such as John Gray, Pierre Proudhon and Karl Marx in Europe, and Josiah Warren in the US [141].

In 1973, the time bank idea was born in the mind of the Japanes visionary Teruko Mizushima, whose objective was to ensure that groups of people exchange work voluntarily, with the principal aim of building a type of social security: when they have time and good health, the users of the time bank offer services to others and when they do not, they consume services from others [71]. In the US, the idea was popularised (and the terms "Time

Bank" and "Time Credit" trademarked!) by Edgar Cahn from 1980. Cahn went on to found the "Timebanks USA" organisation in 1995 [141]. In 1983, the term LETS (Local Exchange Trading Systems) was coined to describe the related idea of local currencies for the exchange of good and services [136].

From the beginning of this century, time banks have been moving into the digital world. At present, the most well-known time-banking web application hOurWorld [55] claims to have 32,475 members in 491 communities and to have handled 1,620,826 hours of service.

In the case of P2P technologies, the first communications between machines for content exchange were carried out from 1978 under UUCP (Unix to Unix Copy) applications and protocols [127], which allow information to be copied from one Unix computer to another through the network. Usenet [142] was born as a subsequent development to UUCP. In 1979, this system, created by Tom Truscott and Jim Ellis, allowed hierarchically-organised articles to be sent between network nodes using a "flooding" mechanism, that is, with a descentralised control structure, for which reason it can be considered a P2P network.

In 1983, Paul Mockapetris, designed the Domain Name System (DNS), a hierarchical naming system for computers, services, or any resources connected to the Internet or a private network, whose servers exchange information between themselves in a descentralised manner, for which reason the DNS servers can be considered to form a P2P network [129].

However, P2P technology became popularly known in 1999, the year in which the first application for music file exchange (Napster) begin its activity on the network [137]. Today, there are millions of users of P2P applications, and everyday P2P applications are used for file exchange, However, to our knowledge, there is no known implementation of a time bank that uses P2P technology for its operation.

2.2. Motivation

Current time bank implementations suffer from the following problems:

- High maintenance costs. To give 24/7 support for a centralized application requires capital to pay for server maintenance and user administration. Since time banks are usually operated by nonprofit organizations that attempt to provide a free service to their users, these costs are hardly bearable.

- Upgrade and adaptability issues. Many existing solutions are implemented in closed source or under copyright licenses. This makes it difficult for existing software to be updated and evolve rapidly to adapt to changing circumstances. Moreover, most of the current open source current solutions are built on weak bases and are not very extensible.

- Registration and access difficulties. User registration requires ensuring that the user is a real person. This involves checking private data and therefore requires a manual process, in which the person wishing to register has to be verified, which is inevitably very slow for large numbers of people. Access to the system depends on the registration so before registering, it is not possible to check whether the system is reliable or not.

- Weak security Although it's obvious that with a centralized solution security is much more affordable, many current centralized solutions have not devoted much effort to security and protection against fraud, simply assuming that the users are trustworthy and well-behaved.

The aim of this project is to set up the foundations that provide an effective solution to all of these problems.

2.3. Goals

The main idea of this project is to use current P2P technologies to specify and design a system and implement a time bank application prototype.

Below is a breakdown of goals for this project:

- **Study and learning of P2P technology.**

- **Specification of a P2P time bank.**

    - **System functionality.**
    - **Use-case diagrams.**
    - **Sequence diagrams.**
    - **Class diagrams.**
    - **System architecture.**
- **Implementation of a P2P time bank prototype.**
- **Simulation of a P2P time bank prototype.**

2.4. Work plan

We spent the first few months defining the project idea and drawing up the first version of the system specification. Once that was completed, we looked for a P2P tool capable of simulating our system in order to obtain results which could be studied.

After experiencing difficulties in understanding the P2P simulators, we undertook a more detailed study of P2P simulation, in particular, and P2P technology, in general, using books, articles and other content on the network. As a result of these difficulties the project was re-oriented more towards the implementation of a prototype.

In the following months, we researched modelling and development tools for model on which to develop the prototype, this development taking place under a flexible use of an agile methodology.

Finally, during the last few months we defined the specification that was to evolve throughout the project and the implementation of part of it in a prototype.

2.5. Scope

The entire contents of the project will be available for free, as free software, on the Internet. Any developer or group of developers interested in

creating a P2P solution for a time bank could then do so by continuing from the base that we have developed in this project.

Capítulo 3

Estado del arte

A lo largo de los últimos años ha ido aumentando el número de aplicaciones que funcionan bajo sistemas P2P. Los sistemas P2P son sistemas distribuidos. Un sistema distribuido es aquel donde sus componentes hardware y software, localizados en ordenadores conectados mediante una red, se comunican y coordinan a través del paso de mensajes [32].

El aumento de las aplicaciones P2P se ha debido, en gran medida, a los importantes avances en las tecnologías de la comunicación, que han reducido notablemente los tiempos de acceso a la información compartida de la red. A lo anterior se suma el objetivo de paliar las deficiencias de los servicios centralizados (implican un importante costo económico en el mantenimiento de los servidores), para lo cual se han diseñado nuevas arquitecturas y protocolos P2P, que han desencadenado muchas batallas legales por temas de derechos de autor sobre la información compartida.

Por otra parte, los bancos de tiempo han empezado a hacerse un hueco en los últimos años en el entorno digital, a pesar de que se idearon en los años ochenta [1]. Algunos piensan que se debe como consecuencia directa a las crisis económicas de los últimos años, otros creen que fue por la entrada en la era digital y otros piensan que se debe a la necesidad de establecer un sistema económico complementario en paises subdesarrollados. Fuese por la razón que fuese, los bancos de tiempo son a día de hoy una realidad en el mundo digital [20].

Pero al tratarse de un sistema que funciona de forma libre y gratuita, las

soluciones software que existen en la actualidad tienen dificultades para costear el mantenimiento del mismo. En este proyecto se pretende dar solución a esta problemática fundamental y al resto, de forma que en el futuro sea posible crear un banco de tiempo gratuito, eficiente, autosostenible y con un nivel de seguridad aceptable.

3.1. Banco de tiempo

3.1.1. ¿Qué es?

Un banco de tiempo es un sistema de intercambio de servicios por tiempo. Es decir, la unidad de intercambio no es el dinero sino una unidad de tiempo (horas, minutos, segundos...).

Su finalidad es proporcionar incentivos y recompensas a aquellos trabajos los cuales un sistema de mercado puro devalúa como por ejemplo el cuidado de ancianos o la enseñanza a niños. Además de ésto, los participantes pueden obtener confianza, contacto social y habilidades gracias a la entrega hacia los demas [141]. Por lo tanto, las comunidades usan los bancos de tiempo como herramienta para forjar conexiones intra-comunitarias más fuertes, contruyen redes sociales de gente que dan y reciben apoyo entre sí para formar conexiones y amistades, mediante el uso de los bienes y recursos que que existen dentro de una comunidad o un grupo [119]. Asimismo los bancos de tiempo pueden verse como mecanismos para lograr coproducción [119].

3.1.2. ¿Cómo funciona?

El tiempo, como moneda de intercambio, se genera a través de crédito mútuo, es decir, cada transacción se registra como un crédito y débito en las cuentas de los usuarios.

En un banco de tiempo, el valor del crédito de cada usuario se mide exclusivamente en las horas realmente trabajadas, independientemente del usuario y de las circunstancias. En la mayoría de los bancos de tiempo las horas de cada usuario valen lo mismo, a menudo se ve este hecho como un principio fundamental de los bancos de tiempo. Sin embargo, también existen bancos

de tiempo que incorporan la idea de multiplicador, que no es más que un coeficiente numérico que se multiplica por las horas que ha ofertado un usuario para calcular las horas que debe pagar el usuario deudor. Esta idea se concibe para hacer una distinción de calidad de los servicios que componen el banco de tiempo, contando el nivel de estudios y experiencia del usuario acreedor y la dificultad del servicio ofrecido o de las diferencias entre el nivel de vida en distintas regiones del planeta (ROCS [110] define tasas de intercambio entre regiones). Teniendo en cuenta este principio, se tienen en cuenta diferentes factores para definir el valor del multiplicador, como pueden ser: la dificultad del trabajo, la reputación en el sistema de la persona que ofrece el servicio o una titulación o certificación que avale que el usuario esté capacitado para impartir el servicio.

A la hora de acordar el valor del multiplicador se podría tener una lista de profesiones (cada una con su valor del multiplicador) pero la definición de tal lista sería muy difícil y su gestión muy costosa, sobre todo en un sistema descentralizado. La implementación más fácil en un sistema descentralizado es que el multiplicador se acuerde libremente entre las dos partes en cada transacción, tal como en el sistema ROCS. Los créditos y débitos de tiempo que se generan en cada transacción se almacenan en el banco de tiempo. Estos créditos pueden ser utilizados en cualquier momento para pagar otros servicios. En algunos países estos créditos y débitos están sujetos a impuestos.

Mientras más transacciones se produzcan en el banco de tiempo, más rica será la comunidad que lo usa, con lo cual **es recomendable incentivar el intercambio de servicios entre los usuarios que componen la comunidad**.

3.1.3. En la actualidad

Como se ha comentado en la introducción de este capítulo, las aplicaciones software que implementan la idea de un banco de tiempo han ido incrementándose a lo largo de los últimos años, tanto en España como en todo el mundo.

Todos tienen una característica en común y es que utilizan un perfil de administrador para gestionar todo lo relacionado con los usuarios (altas, ba-

jas...), además de que disponen de emplazamientos físicos donde se realizan las gestiones. Se pueden destacar los siguientes ejemplos donde se observa esto:

- **hOurworld:** En este banco de tiempo, antes del registro, pide al usuario que se asocie a una comuniadad de intercambio más cercana a él y una vez registrado, se le envía su solicitud a esa comunidad donde se procesan y se comprueban sus datos y, si son correctos, se le envía una respuesta con los siguientes pasos a realizar para poder completar su registro [55].

- **Ayudarnos.org:** Este ejemplo de banco de tiempo presenta un proceso de registro mucho más tedioso que el anterior puesto que presenta, por llamarlo de alguna manera, dos fases de registro. La primera fase es en la que el ususario rellena el formulario web de registro con sus datos personales. La segunda fase es la de validación, en la que se le obliga al usuario a que se dirija a la sede del banco de tiempo para comprobar que los datos que ha rellendo en el formulario se corresponden con los de su DNI o pasaporte, y así pasar a ser un usuario activo [11].

Cabe destacar que cuando intentamos probar el funcionamiento de estos dos sistemas, nunca pasamos de la fase de registro, ya que del primero no recibimos respuesta alguna, y en el segundo pasamos la primera fase pero no la segunda. Pero no solo ocurrió en estos dos ejemplos sino que también se dio en el resto de bancos de tiempo que probamos, y es que, como norma general, ningún banco de tiempo deja registrarse a un usuario sin antes comprobar su identidad.

Además de los dos bancos de tiempo mencionados existen otros como Community Weaver [24], un software basado en Drupal, en Estados Unidos y uno de los mas usados junto a hOurworld. Además existen software para bancos de tiempo en España como TimeOverflow [112]. En España la Asociación para el Desarrollo de los Bancos de Tiempo, una asociación sin ánimo de lucro, que da apoyo a los bancos de tiempo con el fin mejorar su funcionamiento [8].

Algunos de los bancos de tiempo actuales que se han observado incorporan sistemas de reputación rudimentarios en forma de comentarios sobre un

servicio prestado, es decir un sistema de feedback (retroalimentación).

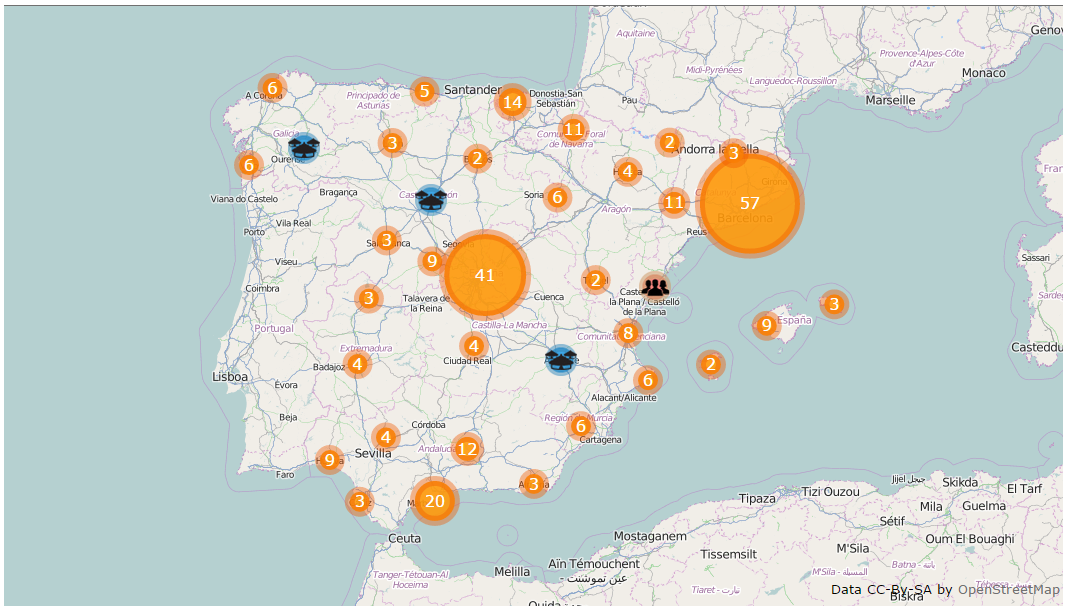

Figura 3.1: Mapa con el número de bancos de tiempo en España según BDT Online [13].

3.1.4. Oxidación

La oxidación de la moneda se refiere a una suerte de tributo, de carácter no recaudatorio, que evita la acumulación. Al estar gravada la tenencia de la moneda, esta circula de forma mucho más rápida evitando la acumulación y fomentando los intercambios [73]. De esta manera se obliga a que circule el dinero dando lugar a que no solo unos pocos acumulen toda la riqueza.

Un ejemplo (de oxidación, "*demurrage*" en inglés) es el famoso experimento de Wörgl en 1932, que demuestra la eficacia de la oxidación en el mundo real. Para más información ver [111].

3.2. Sistema P2P

3.2.1. Arquitectura de los sistemas P2P

Uno de los temas más importantes a tratar a la hora de desarrollar una aplicación P2P es la arquitectura a elegir. En este apartado se exponen los distintas clasificaciones de arquitecturas P2P que se conocen en la actualidad junto a sus ventajas e inconvenientes.

Clasificación según el grado de centralización

Es posible clasificar las diferentes topologías de red P2P en base al número de conexiones que hay entre los pares de nodos de la red. A continuación se explicará en detalle las tres posibles variantes: centralizadas, descentralizadas e híbridas [84].

Redes P2P centralizadas

Esta variante de topología P2P se caracteriza porque sólo tiene un único nodo en la red que actúa como servidor de enrutamiento. Todas las conexiones de los nodos de la red pasan por dicho servidor.

Su mayor ventaja es la conexión directa que se establece entre un nodo cliente y el servidor para conseguir la dirección del nodo que tiene el contenido que se desea obtener. Esta conexión directa permite acceder a bajo coste a la información del servidor, ya que no es necesario realizar ningún salto por los nodos de la red.

Aunque tiene como desventaja la dependencia total respecto al nodo que actúa como servidor, lo que dificulta la estabilidad y escalabilidad del sistema.

Un ejemplo de este tipo de topología de red es el que se encuentra en la aplicación Napster. Más información en [84].

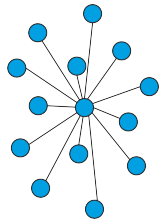

Figura 3.2: Ejemplo gráfico de red P2P centralizada.

Redes P2P descentralizadas En contraposición con la variante anterior, esta topología de red P2P no tiene un único servidor, sino que todos los nodos de la red P2P actúan como cliente y servidor al mismo tiempo.

Esta topología de red tiene muchas ventajas (robustez, estabilidad, escalabilidad...), aunque la más interesante es la robustez que ofrece, ya que mantiene el contenido de la red a pesar de la caídas que pudieran ocurrir en alguno de los nodos que actuasen como servidores.

El mayor inconveniente de esta topología de red es la excesiva cantidad de mensajes que se transmiten por la red, provocando un fenómeno en la red conocido como inundación.

Un ejemplo de este tipo de topología de red es el que se encuentra en la aplicación Gnutella. Para más detalles consultar [84].

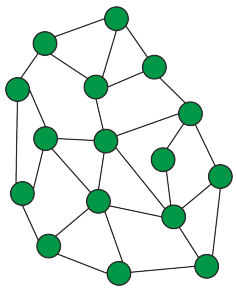

Figura 3.3: Ejemplo gráfico de red P2P descentralizada.

Redes P2P híbridas

Esta topología de red es una combinación de las dos anteriores. En este caso se dispone de un servidor central que redirige a los nodos clientes hacia nodos de la red que actúan como servidores de enrutamiento hacia la información.

Consiste en buscar lo mejor de cada una de las topologías anteriores para combinarlas y establecer cierto equilibrio en la red, de manera que ni todos los nodos son servidores ni se depende de un único servidor por el que pasan todas las conexiones entre nodos de la red.

Muchas de la aplicaciones P2P que se usan hoy en día funcionan bajo esta topología, como por ejemplo Bitstorm [16].

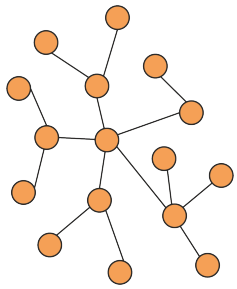

Figura 3.4: Ejemplo gráfico de red P2P híbrida.

Clasificación según la estructuración

La red de sobrecapa (overlay network) es una capa que forma una topología de red virtual sobre una o más redes existentes, que interactúa directamente con los usuarios. Está compuesta por todos los nodos de la red y su objetivo es proporcionar unos servicios de red que no están disponibles en la red/es subyacente/s. Este tipo de redes son utilizadas en los sistemas P2P.

Basándonos en cómo se enlazan los nodos de la red de sobrecapa, las redes P2P se pueden clasificar como estructuradas y no estructuradas.

Red P2P no estructurada

Los primeros sistemas P2P fueron los no estructurados donde los nodos eran totalmente autónomos. Normalmente cada nodo almacena sus propios datos y unos enlaces a otros nodos, llamados nodos vecinos, formando así el esqueleto del sistema. Cuando un nodo quiere unirse al sistema, simplemente contacta con un nodo existente que pertenezca al sistema y hace una copia de sus enlaces. Estos serán utilizados y mantenidos por el nuevo nodo, al margen del nodo original.

La principal ventaja de este tipo de sistemas P2P es que permiten una alta autonomía en los nodos y por lo tanto incurren en un bajo coste de mantenimiento. Sin embargo, esta ventaja deriva en un alto coste en la búsqueda de información, ya que los nodos no tienen un conocimiento global de la distribución de la información en el sistema. Las búsqueda o peticiones deben inundar (flooding) toda la red. Por lo tanto, la escalabilidad de un sistema P2P no estructurado se convierte en un problema preocupante. Como medida para disminuir este problema, todos los sistemas P2P no estructurados incorporan un tiempo de vida (TTL) a las búsquedas de tal forma que el sistema detiene las búsquedas cuando el número total de niveles o pasos en la inundación es igual a la limitación impuesta por el TTL. Por tanto, no se puede garantizar que si existe un resultado para la búsqueda realizada, ese resultado será encontrado, es decir, en la búsqueda puede haber "falsos negativos". Para más detalles ver [84].

Un ejemplo de este tipo de redes son las primeras versiones de Gnutella.

Una red no estructurada puede organizarse de forma jerárquica [53], convirtiendose en una red heterogénea que combina lo mejor de las redes centralizadas y descentralizadas. Algunos de los nodos de estas redes se convierten en supernodos, que son capaces de hacer de "mini servidores" a los que el resto de nodos consultan en busca de la información. Los nodos normales sólo se conectan con los supernodos, siendo ellos los que se encargan de enviar las consultas a otros supernodos de la red. Las ventajas de estas redes es que son más robustas que las soluciones totalmente centralizadas y más rápidas obteniedo la información que los sistemas P2P puros. Por otro lado como inconveniente nos encontramos con la necesidad de un algoritmo que permita elegir los supernodos de la red.

Este tipo de redes son muy comunes en la actualidad. Como ejemplos de redes P2P heterogéneas no estructuradas podemos nombrar a eDonkey, Skype y versiones más recientes de Gnutella.

Red P2P estructurada

Aunque los sistemas P2P no estructurados son los más populares por su simpleza y por ser fáciles de construir, tienen una baja eficacia, por ejemplo, estos sistemas no garantizan un resultado para una búsqueda de información incluso aunque dicha información exista en el sistema. Este problema fue el detonante de la creación de los sistemas P2P estructurados.

Al contrario que en los sistemas P2P no estructurados, los nodos participantes en los sistemas P2P estructurados están obligados a organizarse en alguna topología fija, tales como:

- Anillos (p.ej. Chord [105])
- Hipercubos (p.ej. CAN [94])
- Árboles binarios (p.ej. Kademlia [70])
- Árboles PRR / Mallas (p.ej. Pastry [97], Tapastry [146])
- Listas múltiples enlazadas (p.ej. Skip Graph [9])

Esto quiere decir que en un sistema P2P estructurado, cuando un nodo quiere unirse a la red, debe seguir algunos procedimientos concretos para establecer su posición en el sistema en función del tipo de topología que adopte el sistema. La principal ventaja de este requisito es que al estar basado en una topología fija, el sistema es capaz de organizar la información en un determinado orden para que pueda ser encontrada más facilmente después. Como resultado el sistema puede ofrecer una búsqueda eficiente y efectiva. En particular, se puede garantizar que si existe un resultado para la búsqueda realizada, ese resultado será encontrado, es decir, en la búsqueda no puede haber “falsos negativos”.

Por otra parte, la mayoría de los sistemas P2P estructurados pueden dar una respuesta a una búsqueda en un plazo de O(log N) pasos donde N es el número de nodos en el sistema y cada paso tiene exactamente un mensaje.

Este resultado implica que los problemas de búsqueda en los sistemas P2P no estructurados quedan totalmente resueltos en los P2P estructurados. A pesar de eso, la desventaja de estos sistemas es que la necesidad de una topología requiere un alto coste de mantenimiento. En general, podemos clasificar los sistemas P2P estructurados existentes en tres grandes categorías basadas en la estructura de su red de sobrecapa:

- Basados en tablas hash distribuidas (DHTs)
- Basados en listas
- Basados en árboles

Para más información consultar [84].

Al igual que en los sistemas P2P no estructurados estas redes pueden organizarse de forma jerárquica dando paso a redes heterogéneas estructuradas [52]. El concepto es el mismo, se introducen diferentes roles entre los nodos, convirtiéndose algunos en supernodos con capacidades especiales sobre la red.

Algunos ejemplos de este tipo de redes DHT jerárquicas (HDHT) son [106]: Adaptive Heterogeneous (AH) Chord (basado en una red de tipo Chord), Brocade, Coral, HIERAS, TOPLUS, Kelips, OneHop, Canon, GTap, Cyclone, Chordella y HONet.

Las bases de datos relacionales y centralizadas tienen graves dificultades para gestionar grandes cantidades de información. Hoy en día, existen grandes plataformas que usan DHT para resolver este problema. Algunas de estas plataformas se basan o implementan un conjunto de técnicas denominadas Dynamo [47]. El uso de estas técnicas proporciona una base de datos distribuida clave-valor de alta disponibilidad y gran escalabilidad:

- Dynamo y S3 de Amazon: Dynamo [47] es un sistema de almacenamiento de clave-valor de alta disponibilidad, basado en una DHT y completamente descentralizado, desarrollado por Amazon [4]. Para conseguir

este nivel de disponibilidad, Dynamo sacrifica consistencia en determinadas situaciones de fallo, siempre pueden existir algunos componentes que fallen en un momento dado. Debido a esto el sistema está construido de forma que trata los fallos como un caso normal sin afectar a la disponibilidad o al rendimiento. Son claves la fiabilidad y el crecimiento continuo, para lo que la plataforma debe ser altamente escalable.

Para cumplir con la fiabilidad y las necesidades de escalabilidad, Amazon ha desarrollado un número de tecnologías de almacenamiento, de los cuales S3 (Simple Storage Service) es probablemente el más conocido.

- Cassandra de Apache: El desarrollo de Cassandra [7] comenzó en Facebook. Al igual que Dynamo, Cassandra es base de datos NoSQL basada en un modelo de almacenamiento clave-valor DHT. Su arquitectura distribuida está basada en una serie de nodos iguales que se comunican con un protocolo P2P con lo que la redundancia es máxima. Cassandra está basada en Dynamo por lo que comparte muchos de sus elementos.
- Voldermort de LinkedIn: Voldemort [121] es un sistema de almacenamiento distribuido clave-valor basado en una DHT usado por Linkedin para conseguir una alta escalabilidad. También toma muchos de sus elementos de Dynamo.

Cabe mencionar que estos sistemas al ser internos de un empresa operan en un entorno de confianza [47].

Sistemas de ficheros

Desde sus comienzos, la mayoría de las aplicaciones P2P se basan en sistemas en los que únicamente se comparten archivos, es decir, se busca información en el sistema y esta información es copiada desde los nodos de la red al nodo que la solicita. Pero existen muchos otros sistemas que requieren la modificación de archivos. ¿Qué problemas presenta este punto de vista para la tecnología P2P?, ¿Es fácil implementar este tipo de sistemas en P2P?.

Se puede deducir rápidamente que en un sistema P2P donde existe una alta redundancia puede ser muy complicado tener todas las copias de los

archivos actualizadas si se realiza una modificación en uno de ellos. Incluso, debido al *churn* (entrada y salida de nodos en la red), puede que alguno de los nodos que tenga copias de archivos que se están modificando esté en ese momento apagado. ¿Qué ocurre cuando este nodo se enciende y se une a la red?.

Existen multitud de estudios que han tratado esta problemática [21, 6, 51], y aunque se pueden encontrar algunos sistemas que pretenden crear una capa encima de un P2P estructurado como PAST [38] (PASTA [74]) o Chord [105] (CFS [28]) que actúe como un sistema de ficheros o base de datos, todavía están iniciándose y son poco sólidos, lo que dificulta mucho trabajar con ellos. Obviamente, si existiera un sistema de este tipo bien trabajado y estable, el trabajo en este proyecto sería mucho más fácil.

Existen trabajos más recientes que tienen como objetivo contruir un PaaS (Platform As A Service) de tipo *cloud computing* encima de una subestructura de red P2P. Mas información en [69, 12].

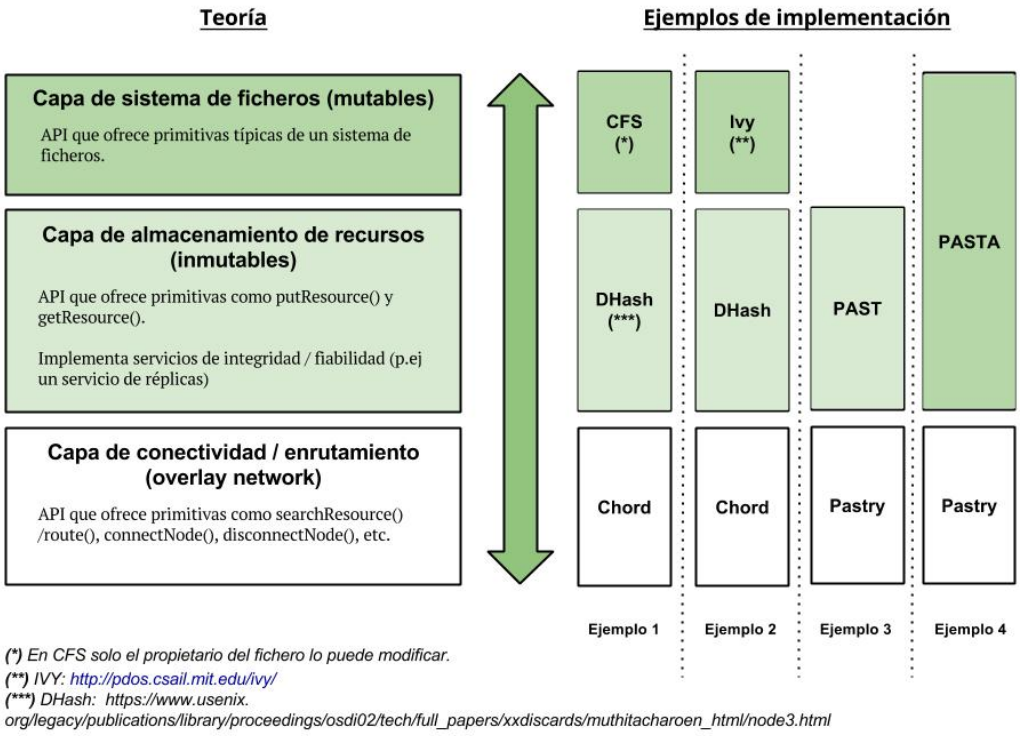

Figura 3.5: Sistema de ficheros basado en una red P2P estructurada.

Confianza y sistemas de reputación

La confianza

La confianza se puede ver en muchos ámbitos del día a día. En particular, cuando alguien solicita un servicio espera recibirlo de la mejor manera posible. La confianza puede definirse como la creencia o confianza en la honestidad, la bondad, la habilidad, o la seguridad de una persona, organización o cosa.

En general, hablaremos de dos categorías dentro de los modelos de confianza: los basados en credenciales (el más conocido es el sistema de clave pública y clave privada, del que se habla en la sección de criptografía) y los basados en reputación, donde cada participante tiene una puntuación basada en lo que ha hecho en el sistema. La primera opción no está muy adaptada al mundo P2P, salvo si se utiliza certificación por la comunidad o partes de la comunidad en un enfoque del tipo "web de confianza" de PGP [147]. Por tanto, se focalizará en confianza por reputación.

Los modelos de confianza basados en reputación se usan normalmente cuando se necesita saber la calidad de lo que una persona está ofreciendo en base a otras ofertas anteriores y no por una credencial (titulación) que asegura que la persona es apta para llevar a cabo una actividad. Esta distinción es importante ya que en muchos casos los servicios de una persona pueden ser buenos en el momento de su registro en el sistema, pero con el paso del tiempo pueden ir empeorando.

A continuación se mostrarán algunos de los modelos de confianza basados en reputación más relevantes para la aplicación desarrollada en este proyecto.

- **Sistemas de búsquedas de reputación basadas en el protocolo tipo gossiping**. La búsqueda de reputación es parecida a la búsqueda de recursos en una red P2P no estructurada (o gossiping protocols [132]). Algunos ejemplos son:

  - **P2PRep**. En P2PRep [26] la confianza de un nodo se determina mediante un proceso de votación de nodos existentes en el sistema que han hecho transacciones en el pasado con ese nodo. La reputación de un nodo también se almacena localmente en los nodos

que han hecho transacciones con éste en el pasado.

Existen dos versiones de P2PRep: la de votación básica y la de votación mejorada. La única diferencia entre ambas es que la primera versión trata todas las opiniones de reputación sobre un nodo por igual mientras que en la segunda además considera la credibilidad de las opiniones. La credibilidad aumenta si el resultado de la transacción es consistente con la opinión del nodo y decrementa en caso contrario. De esta forma, se le da mayor importancia a las opiniones de los nodos más creíbles.

- **XRep**. Sucesor de P2PRep. Similar a este excepto en que no sólo considera la reputación de un nodo sino también la reputación de sus recursos. De ésta forma no solo es más fácil seguir la pista a los usuarios maliciosos sino también los recursos dudosos [29].

- **Sistemas que calculan la confianza a través de grafos de confianza.** Calculan la confianza en función de caminos entre nodos a través de grafos de confianza. Un ejemplo es Cooperative Peer Groups [67], que es un sistema de modelo de confianza contruido en lo alto de NICE [75], una plataforma para implementar aplicaciones de cooperación en Internet. El sistema identifica buenos compañeros para formar grupos cooperativos y aísla compañeros maliciosos. En este sistema, después de cada transacción recibida, el nodo valora la calidad de ésta y la puntuación la almacena localmente en forma de *cookie*. La confianza se determina mediante la búsqueda del camino más sólido, en el grafo de confianza, formado a través de las relaciones entre los nodos que han tenido transacciones exitosas entre sí. Otro ejemplo de sistema que utiliza grafos de confianza es Ripple [96].

- **Sistemas que hacen cálculos mas complicados.** Un ejemplo de estos sistemas es PeerTrust [145]. PeerTrust analiza la reputación de un nodo teniendo en cuenta más detalles. Señala que si la reputación de un nodo está basada simplemente en el número de transacciones exitosas, puede no ser suficiente para eliminar nodos maliciosos.
  Como resultado, PeerTrust sugiere cinco factores importantes que se deberían usar para eliminar los nodos maliciosos. Estos factores son: cantidad de transacciones satisfactorias, número de transacciones, credibilidad del *feedback*, contexto de la transacción, contexto de la comu-

nidad.

La gestión de la confianza se refiere a como se almacena la información relativa a la reputación y somo se recupera. Puede ser centralizada, descentralizada tipo P2P no estructurado y descentralizada tipo P2P estructurado.

- **Centralizada.** Un servidor almacena la reputación de cada nodo en la red y la gestiona. Un ejemplo de gestión de confianza P2P centralizada es XenoTrust [35].
- **Descentralizada tipo P2P no estructurado.** La descentralizada tipo P2P no estructurado usa el algoritmo gossiping para el intercambio de información entre los nodo del sistema. Cada nodo envía la puntuación a todos los nodo que conoce. Este método es muy costoso dado que requiere un almacenamiento global de la información de todos los nodo del sistema en cada nodo. Alternativamente, muchos sistemas sugieren que un nodo únicamente debería mantener la información de los nodos que han hecho transacciones con él. Un ejemplo de gestión de confianza tipo P2P no estructurado es EigenRep [64].
- **Descentralizada tipo P2P estructurado.** La descentralizada tipo P2P estructurada indexa la reputación de cada nodo en la red a través de un identificador. Cuando un nodo desea recuperar la reputación de otro, envía una consulta a través de la red con el identificador del nodo que quiere encontrar. Un problema potencial es que si un nodo puede guardar la reputación de sí mismo y si está a cargo del rango de valores de su identificador sería posible que pudiera cambiar los valores relativos a su reputación. Para evitar este problema, el sistema puede replicar la reputación de un nodo en varios nodos. Sin embargo, si hay muchos nodo maliciosos, y cooperan entre ellos, todavía sería posible que pudieran proporcionar valores negativos de algunos nodo. La solución a esto es preguntar a un nodo para evaluar también la reputación de los nodo referenciados. Esto puede ser costoso, pero en muchos casos el sistema puede decidir la veracidad de la reputación en pocos pasos.

  Un ejemplo de gestión de confianza tipo P2P estructurado es P-Grid. P-Grid evalúa la reputación de un nodo en función del número de quejas hechas por éste y el número de quejas sobre éste. P-Grid se basa en la idea de que si un nodo es malicioso siempre recibirá muchas quejas

pero también se quejará de los compañeros que han hecho transacciones con él para defenderse de estas quejas.

La información almacenada es muy básica: pares de quejas (quejas hechas por un nodo y quejas hechas contra ese nodo) que se almacenan en el mismo sitio de la red. La reputación se calcula como el número de quejas contra un nodo multiplcado por el número de quejas hechas por un nodo. Este cálculo nunca se almacena en la red. Más información en [83].

Sistemas de reputación

Es imprescindible generar una confianza, una garantía en los usuarios de que lo que van a solicitar es real. Hoy en dia, la gente todavía es muy reacia a llevar a cabo operaciones por Internet que tengan algún coste sin ningún tipo de referencia.

Existen muchos sistemas de reputación básicos, basados en las valoraciones que hacen otros usuarios del sistema en base a la experiencia que han tenido. Este tipo de sistema de reputación se puede encontrar en multitud de plataformas muy importantes como pueden ser Amazon [4], eBay [39], Slashdot [101], Stack Overflow [104], etc. La clave de este sistema es una valoración mixta que mezcla una puntuación con un sistema de comentarios que los usuario pueden utilizar para crear una valoración más rica.

También hay que tener en cuenta los sistemas de reputación más sofisticados. Después de una investigación no se ha encontrado ningún sistema de reputación automático P2P en el que se alamacenan los valores de reputación, resultado de un cálculo. Sí se han encontrado sistemas en los que se almacenan en la red los datos básicos a partir de los cuales se hará el cálculo. En estos sistemas, cada vez que un nodo está interesado en saber la reputación de otro nodo, tiene que recoger la información básica de la red (un proceso que puede ser costoso, sobre todo si el cálculo implica una dimensión social) y volver a hacer el cálculo. Esto es debido a la problemática de decidir en quién se podría confiar para poner al dia el cálculo después de cada transacción. Sin embargo, otra posibilidad sería un sistema de reputación P2P que funcione mediante un tercer nodo (o incluso más) que actúe de jurado.

3.2.2. Ventajas e inconvenientes de usar un sistema P2P

Gracias al uso de las tecnologías P2P se divide el mantenimiento del sistema entre los computadores (nodos) de la red, este hecho tiene las siguientes ventajas e inconvenientes:

- Ventajas:
  - Reduce enormemente los gastos de administración y los que hay, se distribuyen entre los nodo de la red.
  - Aumenta enormemente la robustez y la tolerancia a fallos.
  - Mejora la capacidad de acceso a la información (réplicas de la información en distintos nodos de la red).
  - Dificulta el control del sistema por una sola entidad. Para el usuario es un sistema con mayor privacidad.
  - Proporciona escalabilidad a bajo coste. Los recursos que se necesitan cuando aumenta el número de usuarios los aportan los propios usuarios.
- Inconvenientes:
  - La administración es muy compleja. **Solución.** No hace falta administrar el sistema global y cada usuario administra su participación en él.
  - Los nodos individuales no siempre están disponibles y su disponibilidad es impredecible (*churn*), lo que dificulta la tarea de asegurar la disponibilidad de los datos.
    Solución. Se dispone de un sistema que tiene entrada y salida ordenada de los nodos, de manera que si un nodo abandona la red su información pasa a otro antes de que este la abandone, y si un nodo entra en la red recibirá aquellos recursos que le sean asignados. Además en el sistema habrá un número suficiente de réplicas de cada recurso alojada cada una en un nodo distinto para que sea muy improbable que todos estos nodos estén caídos (es decir, han salido de manera no ordernada) a la vez.

- Es difícil gestionar la persistencia y no persistencia, también gestionar recursos mutables en presencia de *churn* y de réplicas.
  Solución. No permitir la mutabilidad. En un sistema donde todos los recursos son inmutables, una solución al problema de la acumulación de recursos no deseados que no se pueden borrar, es hacer que los recursos caduquen después de cierto tiempo (recolección de basura distribuida).
- Es complicado conseguir un nivel aceptable de seguridad y de confianza (es difícil evitar, incluso detectar, el fraude).
  Solución. El sistema puede usar hashing y criptografía asimétrica (con firmas digitales) además de usar réplicas. Con el objetivo de asegurar que la información se transmita por la red sin posibilidad de que un tercero la modifique. También debería disponer de un sistema de reputación para aumentar la confianza.
- Problemas para que una sola entidad pueda controlar el sistema, ya que desde el punto de vista de una empresa, es un problema si quiere manipular toda la información.
  Solución. Utilizarlo solamente de manera interna en la empresa.
- Dificultad para asegurar la integridad de los datos, ya que pueden estar alojados en cualquier sitio de la red.
  Solución. Utilización de *hashing*, criptografía y validación/certificación de réplicas.
- Menor probabilidad de asegurar la justicia en el uso del sistema, es decir, evitar la presencia de usuarios que usan los recursos de los demás sin contribuir recursos propios. **Solución.** Usar un sistema de incentivos.

3.3. Criptografía

La criptografía es la ciencia de la escritura oculta, un arte muy antiguo que se remonta a los egipcios en el año 1900 A.C. Ocupa técnicas de cifrado o codificado con el objetivo de conseguir confidencialidad en los mensajes, cuando la comunicación se hace sobre un medio que no es de confianza.

Con la aparición de la informática y el uso masivo de las comunicaciones digitales, surgen numerosos problemas de seguridad para los que se ha utili-

zado la criptografía.

Tomando como contexto la comunicación entre dos aplicaciones, existen algunos requisitos de seguridad específicos:

- **Autenticación**: El proceso de probar la identidad de uno mismo.
- **Privacidad/confidencialidad**: Asegurar que nadie puede leer un mensaje excepto su receptor.
- **Integridad**: Asegurar al receptor que el mensaje que recibe no ha sido alterado de ninguna forma.
- **No repudiación**: Mecanismo que prueba que el emisor es quien envía el mensaje.

En este caso la criptografía no solo se usa para proteger la información sino que también es utilizada como método de autenticación de usuarios.

3.3.1. Tipos de algoritmos criptográficos

Existen multitud de formas de clasificar los algoritmos criptográficos. A continuación se explicara la clasificación más general, basada en el número de claves utilizadas, su aplicación y uso, para poder finalizar con el concepto más importante que se utilizará en el sistema que se propone, la firma digital.

- **Criptografía simétrica o de clave secreta (SKC)**: Utiliza una sola clave para la encriptación y la desencriptación.
- **Criptografía asimétrica o de clave pública (PKC)**: Utiliza una clave para la encriptación y otra para la desencriptación.
- **Funciones Hash**: Utiliza una operación matemática irreversible para la encriptación (criptografía en un único sentido).

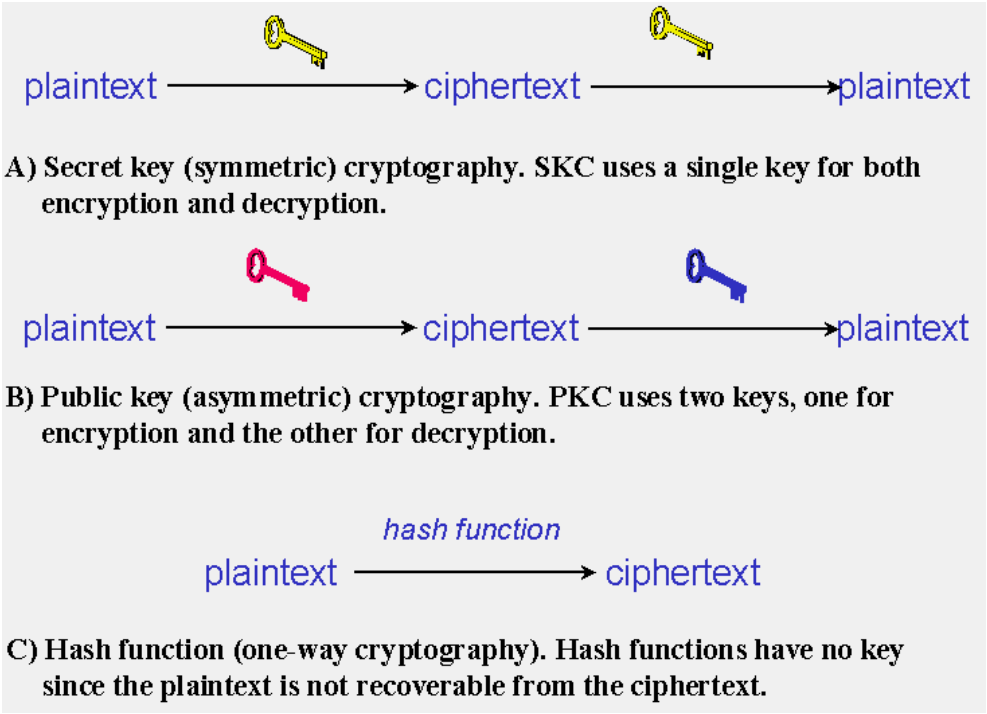

Figura 3.6: Tipos de criptografía.[46]

Llegados a este punto, hay que plantearse una serie de preguntas tales como, ¿por qué existen tantos tipos de algoritmos de encriptación?, ¿no se puede hacer todo lo que se necesita con un método en concreto?.

La respuesta es que cada tipo de algoritmo tiene un uso específico para el que es más adecuado. Las funciones hash, por ejemplo, son usadas para asegurar la integridad ya que cualquier cambio realizado sobre el contenido del mensaje provocaría que el receptor calcule un hash diferente al original. Es muy improbable que dos mensajes generen el mismo hash por lo que la integridad está asegurada a un nivel muy alto.

Por otro lado, la criptografía simétrica es ideal para encriptar mensajes, proporcionando privacidad y confidencialidad. Tanto el emisor como el receptor necesitan la misma clave para encriptar y desencriptar el mensaje.

En cuanto a la criptografía asimétrica, puede usarse para la no repudiación y la autenticación de usuario. Aunque teóricamente también podría utilizarse para la encriptación de mensajes, raramente se usa debido a que la criptografía simétrica es unas 1000 veces más rápida.

También es posible combinar estas funciones para crear un medio de transmisión seguro mediante criptografía híbrida. Con esta criptografía es posible compartir, mediante cifrado asimétrico, una clave simétrica utilizada para la encriptación del mensaje. Compartir una clave simétrica no es una práctica segura, por eso, la clave usada es diferente para cada sesión.

3.3.2. Certificación de claves

La idea es conectar la clave privada o pública con la identidad de la persona física para certificar la autenticidad de que la clave pertenece a su dueño. Por ejemplo, el hecho de descargar una llave pública de algún sitio perteneciente a una asociación no nos asegura que podamos confiar en dicha asociación.

Existen tres maneras de certificar esto:

- **Autoridad de certificación.** Es la entidad que garantiza la autenticidad y veracidad de los datos recogidos en el certificado digital expedido. Se trata de una suerte de institución notarial que ofrece fidelidad a un hecho jurídico [10].

- **Certificación por la comunidad de tipo "web de confianza" de PGP [147].** PGP proveé de un modelo de confianza que ha sido denominado "web de confianza'. Una información que liga un nombre de usuario a una clave puede ser digitalmente firmada por un tercer usuario para dar testimonio a la asociación entre alguien (realmente un nombre de usuario) y la clave. Se usa a la comunidad para crear un "certificado de identidad".

- **Autocertificación.** La clave privada es la identidad.

3.3.3. Firma digital

El objetivo fundamental de la firma digital, es conseguir la no repudiación y la autenticación de usuario, utilizando criptografía asimétrica para su propósito. Imaginemos que un usuario A quiere enviar un mensaje a través de una aplicación a otro usuario B del sistema. ¿Cómo puede estar seguro B de que el mensaje recibido pertenece con total seguridad a A?.

Normalmente la firma digital se construye en dos pasos. Primero, A genera el hash del mensaje; segundo, encripta el valor del hash con su clave privada. Tras la recepción de la firma digital, B recupera el hash calculado por A desencriptando la firma digital con la clave pública de A. De esta forma B puede generar el hash del mensaje recibido y compararlo con el hash obtenido de la firma digital. Si coinciden, B puede estar seguro de que el mensaje no ha sido alterado y su emisor es A.

3.3.4. *Timestamps*

Un sellado temporal (*timestamp*) es una marca de tiempo que garantiza que una determinada información existía en el momento indicado.

El sellado temporal confiable, o trusted timestamping, es una funcionalidad criptográfica con casos de uso más limitados que los de primitivas más comunes como el cifrado y cálculo de hashes. No obstante, pueden ser muy útiles en múltiples ocasiones. A continuación se explicarán brevemente dos tipos de sellados temporal confiable. Para más información ver [58].

Time Stamping Authority (TSA)

La solución más clásica es la basada en *Time Stamping Authorities*. La confianza que se deposita en estas autoridades es similar a la que recae sobre las autoridades de certificación. Lo que hace una TSA es firmar las peticiones de sellado temporal que recibe de sus clientes, incluyendo el instante de tiempo en que se emite la firma. Más información en [58].

Timestamp seguro sin TSA

Con la llegada de Bitcoin [15] han aparecido también servicios que hacen uso de su novedosa infraestructura para crear *timestamps*. En estos servicios, siguiendo el modelo descentralizado de Bitcoin, no es necesario confiar en ninguna autoridad central, sino en las propiedades criptográficas de Bitcoin. Un ejemplo de este tipo de *timestamps* es BTProof [19]. Más detalles en [58].

Capítulo 4

Un banco de tiempo P2P

Un objetivo de este trabajo es hacer un sistema sin administrador, ya que, en muchas ocasiones en las que se requiere un administrador voluntario, al final resulta que no se encarga nadie de gestionarlo. Además, con ésto, se pretende conseguir que el alta en el sistema sea más ágil.

Hay que tener en cuenta que mientras más transacciones se realicen en el banco de tiempo, más se enriquece la sociedad que lo utiliza. Para conseguirlo hay que motivar a los usuarios para que usen sus créditos, con lo cual, como trabajo a futuro se podría estudiar cómo llevar a cabo una oxidación del crédito, es decir, el crédito de cada usuario perderá valor proporcionalmente con el tiempo que lleve sin comprar ningún servicio.

4.1. Soluciones para los inconvenientes

Para los inconvenientes citados en el capítulo anterior, se han encontrado las siguientes soluciones:

- Inconvenientes (con posibles soluciones):
    - La administración es muy compleja.
      Solución. En este banco de tiempo no existiría ningún tipo de administración.
    - Los nodos individuales no siempre están disponibles y su disponibilidad es impredecible (*churn*), lo que dificulta la tarea de asegu-

rar la disponibilidad de los datos.
Solución. Se utilizaría una plataforma P2P que se encargaría de gestionar la salida y entrada ordenada de la red de los nodos y de la utilización de réplicas para tratar las caídas.

- Es difícil gestionar la persistencia y no persistencia, también gestionar recursos mutables en presencia de *churn* y de réplicas.
Solución. En este banco de tiempo los ficheros caducarían no permitiendo la mutabilidad.
- Es complicado conseguir un nivel aceptable de seguridad y de confianza (es difícil evitar, incluso detectar, el fraude).
Solución. En este caso se utilizarían sistemas criptográficos con firmas digitales para garantizar la no repudiación y la autenticación de los usuarios. Así como usar un sistema de reputación para evitar ataques de distinto tipo, en particular ataques Sybil [34].
- Dificultad para asegurar la integridad de los datos, ya que pueden estar alojados en cualquier sitio de la red.
Solución. Utilización de una plataforma P2P y de técnicas de *hashing* y las firmas digitales (cada fichero firmado por su propietario).
- Menos posibilidades de asegurar la justicia en el uso del sistema, es decir, evitar la presencia de usuarios que usan los recursos de los demás sin contribuir recursos propios.
Solución. El banco de tiempo podría utilizar un sistema de oxidación y/o de incentivos.

4.2. Tecnología adecuada

Desde un primer momento, se decidió que lo más conveniente para llevar a cabo el desarrollo de este proyecto era usar una red P2P descentralizada y estructurada para los subsistemas de transacciones y de usuario (pero no necesariamente sería la elección más adecuada para el subsistema de servicios), ya que se quería conseguir una gran eficacia a la hora de buscar los archivos de la red.

La idea inicial de este proyecto era especificar un sistema y mediante una herramienta de simulación P2P obtener unos informes, estadísticas, y

detalles del funcionamiento de este, simulando una red P2P y, si el tiempo lo permitiese, hacer un pequeño prototipo. Investigando los simuladores más relevantes [117, 98] se observó que la mayoría estaban obsoletos y no cumplían con las características que se necesitaban. Fue entonces cuando se decidió enfocar el proyecto más hacia la implementación.

Cabe mencionar que el lenguaje de programación que se pensó para desarrollar la aplicación fue Java, ya que es un lenguaje que más se utilizó a lo largo de la carrera. Además la mayoría de las redes de sobrecapa (overlays network) están implementadas en este lenguaje.

A la hora de elegir el protocolo se decidió optar por Chord, entre otras opciones como Pastry [138] o Kademlia [135], ya que algunos de los integrantes del equipo lo habían visto durante la carrera. Se hicieron algunas pruebas en un chat sencillo construido sobre una implementación open-source de Chord llamado Open Chord [77]. Durante esas pruebas surgieron varios problemas como la lentitud en el envío de mensajes o incluso, a veces, nunca llegaban al destinatario, problemas con la conexión de un nodo a la red, que provocaron dudas acerca de usar esta implementación. Entonces, se decidió dejar de lado Open Chord y probar FreePastry [44] sobre el mismo modelo. Las pruebas usando FreePastry mostraron unos resultados bastante más optimistas de acuerdo con lo que se esperaba ya que los mensajes se enviaban sin que se pudiera apreciar retardo alguno y la conexión de los nodos era mucho más rápida y eficiente. Además la documentación era mejor, aunque seguía siendo pobre.

Por otro lado, no queríamos utilizar el block chain de Bitcoin por el gasto de energía que supone la "minería.en la que se basa su seguridad, así como el hecho de que cada transacción Bitcoin tiene que gastar Bitcoins (aún si el gasto mínimo, de momento, es muy bajo).

4.3. Estructura de la aplicación

Este banco de tiempo va a estar compuesto de cuatro subsistemas principales bien diferenciados. El primer subsistema se encargará de la gestión de los usuarios, dicho subsistema se definirá como subsistema básico. El segundo

subsistema se encargará de gestionar las transacciones. El tercer subsistema será el encargado de la gestión de los servicios, y el último subsistema de las notificaciones. A continuación se explicarán brevemente cada uno de los subsistemas.

- **Subsistema de gestión de usuarios o básico** Los usuarios son la parte fundamental del sistema. En este módulo se gestionan todas las acciones que afectan a los propios usuarios.

- **Subsistema de gestión de transacciones** El servicio es el producto principal entorno al que gira el banco de tiempo. Es necesario que se puedan realizar intercambios adecuados, es decir, permitir dar o recibir tiempo. Este subsistema existe para gestionar las transacciones / movimientos de las distintas operaciones que se den entre los usuarios del banco de tiempo.

  Cuando dos usuarios quieren ponerse en contacto para formalizar un servicio ambas partes deben estar de acuerdo. En este sistema el contrato se divide en varias etapas que se van completando a medida que avanza el proceso de comunicación entre los dos usuarios. Mediante esta comunicación se consigue llevar a cabo un servicio.

  En el caso general, las transacciones pueden ser de tipo uno a uno, uno a muchos (por ejemplo, una clase con un profesor y varios alumnos) o muchos a muchos (por ejemplo, una clase con varios profesores y varios alumnos). Un posible procedimiento para tratar estos casos es el siguiente: primero, decidir un valor para el multiplicador, segundo sumar las horas de los acreedores (los múltiples profesores en el ejemplo), tercero dividir las horas resultantes entre los deudores (los múltiples alumnos en el ejemplo). En este proyecto, sin embargo, solo se contempla transacciones de tipo uno a uno, tanto en el análisis (el diagrama de dominio del Apéndice B.2 solo considera transacciones de tipo uno a uno) como en la implementación del prototipo.

  Además, un servicio puede estar compuesto de múltiples transacciones, por ejemplo el servicio de dar clases de piano, y la relación puede, o no, estar formalizada en un contrato. Por otro lado, además del pago, cada

transacción puede incluir un prepago que consta de un presupuesto, una factura proforma y una factura. En este proyecto, estas distintas posibilidades están recogidas en el análisis (ver diagrama del dominio del Apéndice B.2) pero el prototipo solo contempla servicios compuestos de una solo transacción y siempre con un prepago (presupuesto, factura proforma y factura).

En el subsistema de transacciones, se supone que cada parte empieza la transacción en posesión del hash del perfil y de la clave pública de la otra parte. Esta información se consigue en el subsistema de gestión de servicios que se ejecuta previamente.

Se puede entender que un banco de tiempo no deja de ser un banco tradicional donde se utiliza otro tipo de recurso (tiempo) para los intercambios y operaciones de los usuarios.

- **Subsistema de gestión de servicios** Los servicios generan la riqueza del banco de tiempo. En este subsistema un usuario tiene la posibilidad de ofertar, demandar y/o buscar un servicio.

  Un usuario que tiene unos servicios ofertados tiene dos opciones:

  - Esperar a que otros usuarios interesados contacten con él.
  - Contactar con usuarios que demanden servicios como los suyos.

  Se especificó en la primera iteración del proyecto, la primera de las opciones por considerarse más prioritaria.

- **Subsistema de gestión de notificaciones** Las notificaciones son imprescindibles para la comunicación en el sistema. Se podrían utilizar varios sistemas para gestionar las notificaciones. En la primera iteración se optó por usar un sistema de correo electrónico pero también se podrían haber utilizado comunicaciones móviles, incluso que funcionen bajo P2P como Bleep [18].

En un primer momento nuestro objetivo era cubrir todos los módulos, aunque finalmente nos terminamos centrando en los subsistemas de gestión de usuarios y de gestión de transacciones.

4.4. Gestión de los riesgos de funcionamiento

La idea sería llevar a cabo una aplicación de banco de tiempo P2P escalable y automática, por consiguiente, es de vital importancia que incorpore un sistema de reputación como medida para intentar evitar el fraude entre los usuarios de la aplicación. A continuación se proponen una serie de gestiones que se incluirían en la aplicación:

4.4.1. Límites de crédito o débito

Con el fin de dificultar el fraude, se podrían establecer límites de crédito y débito:

- **Limite de crédito diario de un usuario.** . Lo normal sería poner un límite de ocho horas al día; si se utiliza un multiplicador, el límite lo tendría que tener en cuenta.

- **Límite de crédito absoluto de un usuario.** Límite del total de horas de crédito que pueden ser acumuladas; este límite debe ser proporcional a la reputación de los clientes del usuario.

- **Límite de débito absoluto de un usuario.** Límite el total de horas de débito que pueden ser acumuladas; este límite debe ser proporcional a la reputación del usuario. Desprestigiar a los "gorrones".

Estos límites reducen el fraude. Por ejemplo, el límite de crédito debe reducir cualquier ganancia de un usuario malicioso que acumula crédito mediante la realización de servicios ficticios para usuarios ficticios. Sin embargo, tambien se debe configurar el sistema de reputación para que este usuario malicioso no pueda adquirir reputación a partir de la realización de esos trabajos.

4.4.2. Sistema de reputación básica

El sistema de reputación básico consta de una evaluación numérica y un comentario sobre el comportamiento de la otra parte en cada transacción es el más fácil implementar.

En el caso de una aplicación P2P, como medida de seguridad, se podría introducir redundancia, duplicando la información sobre reputación mantenida por el sistema:

- Por un lado, el sistema contiene un fichero de comentarios sobre este usuario proporcionados por otros usuarios con los que ha tenido alguna transacción.
- Por otro lado, el sistema contiene un fichero de comentarios proporcionados por este usuario sobre otros usuarios con los que ha tenido alguna transacción.

Esto se incluye en la especificación del banco de tiempo como *Feedback About Me* y *Feedback By Me*.

4.4.3. Sistema de reputación más sofisticada

Cálculo de la reputación

La reputación podría puntuarse en una escala desde un valor máximo (MAX) hasta un valor mínimo (-MAX).

Un grupo inicial de confianza, por ejemplo, los fundadores del sistema, quienes deberían ser identificados por todos los usuarios, empezarían con una reputación positiva (por ejemplo, MAX/2) con el fin de arrancar (dar comienzo) al sistema de reptutación. El resto de los usuarios empezarían con una reputación de 0. Esto quiere decir que la reputación solo puede obtenerse directamente o indirectamente mediante interacción con un grupo inicial de confianza. Pero otras soluciones son más vulnerables a ataques Sybil [124].

Aumento de la reputación

Una transacción exitosa aumenta la reputación en X, multiplicada por los siguientes factores:

- La opinión de la otra parte acerca de la transacción. Por ejemplo, escala 1 a 10, valor por defecto, 5.

- La credibilidad de la opinión de la otra parte.
  - Esto podría ser simplemente una función de la reputación de la otra parte; también podría ser algo más complicado, por ejemplo, ponderado por la desviación entre las opiniones de la otra parte sobre usuarios con los que ha tenido transacciones con anterioridad y la "opinión general" calculada sobre esos mismos usuarios (así que una opinión sobre otro usuario es más creíble si las opiniones previas del usuario son compartidas por muchos otros usuarios). Este enfoque, usado en el sistema PeerTrust, ayuda a evitar la colusión (confabulación), incluyendo ataques Sybil basados en colusión en grupo.
  - Se podría añadir también un sistema basado en confianza por certificación de la comunidad (tipo "web de confianza"), sin que sea obligatorio. Simplemente sería un dato más de confianza y quizás una razón para tener límites de crédito y débito más altos.
  - Si la reputación de la otra parte es menor o igual que 0, la transacción no puede resultar en un aumento de la reputación, es decir las opiniones de nuevos usuarios sobre otros no tienen valor. Nótese que aunque esta política dificulta los ataques, no los evita y que un usuario que tiene una reputación positiva puede hacer trabajos para usuarios nuevos ficticios creados por él (ataque Sybil) por darles reputación. Por tanto, sería más útil si se utilizará conjuntamente con un enfoque similar al de PeerTrust mencionado anteriormente.
- El tamaño de la transacción, es decir el número de horas implicadas.
  - Esto se puede obtener de la información de la transacción.
  - Cuanto mayor es la transacción, mayor reputación otorga a sus participantes; esto es para asegurar que la reputación del sistema no sea parcial en favor de muchas transacciones pequeñas y para asegurar que el comportamiento correcto en muchas transacciones pequeñas pero maliciosas en grandes transacciones no sea una estrategia ganadora.
- Si la parte en cuestión dio un comentario o no.

- Por ejemplo multiplicar la cantidad dada por 0.8 si no se proporcionó un comentario. De este modo se fomentan los comentarios.

También se podría aumentar automáticamente la reputación de un usuario si en un período de tiempo dado (un tiempo relativamente largo), ha participado en al menos un cierto número de transacciones sin recibir una opinión negativa (debajo de cierto umbral) en ninguna de ellas.

Disminución de la reputación

La reputación disminuye en una cierta cantidad si, un tiempo después de que el deudor recibe la factura de un servicio, la transacción no se ha completado. La disminución podría llevarse a cabo cuando una transacción se aborta (después de haber consumido el número permitido de reintentos).

Se podría considerar también si abortar una transacción en otro punto del protocolo de pago también debe llevar a la pérdida de reputación.

La reputación también debe disminuir si la cuenta de un usuario no tiene crédito durante mucho tiempo.

Otra forma de reducir la reputación sería si un usuario alcanza su límite de endeudamiento.

Gestión de la reputación

La gestión de la reputación se refiere a la modificación, almacenamiento y recuperación de la información relativa a la reputación. Se puede confiar en un usuario para almacenar, ya sea localmente o en la red, información de reputación sobre otros usuarios pero no se puede confiar en él para calcular su propia reputación o la de los demás basada en esa información, después de una transacción en la cual estaba implicado. Duplicando el cálculo en la otra parte no evitaría los problemas ya que los dos usuarios podrían coludir. Por tanto, tenemos las siguientes posibles soluciones:

- Usar una o varias terceras partes de confianza: en nuestro caso, la tercera parte que llevará a cabo la tarea del cálculo después de cada transacción tendría que ser seleccionada aleatoriamente del conjunto de usuarios no implicados en la transacción o, quizá, que no estén implicados en ninguna transacción con los usuarios en cuestión (si es posible).
- Almacenar solamente información de reputación en bruto en la red y el cálculo de la reputación de un usuario B a partir de esta información en bruto debe hacerse por un usuario A, interesado en contratar los servicios del usuario B. Esto podría hacerse por A, recuperando primero la información sobre la última transacción de B y luego siguiendo los enlaces necesarios para obtener toda la información en bruto necesaria.

Finalmente, se podría usar una caché para evitar una cierta cantidad del cálculo, como se hace en PeerTrust.

Sobre lo que se ha especificado *Feedback About Me* y *Feedback By Me*, cabe mencionar cómo la duplicación del *Feedback* da algo de seguridad pero solo si no hay colusión entre las dos partes.

Por falta de tiempo, en el prototipo no se ha implementado el sistema de reputación más sofisticado (menos la valoración numérica sobre la otra parte que ya forma parte del sistema de reputación básica).

4.4.4. Sistemas de incentivos

Oxidación de la reputación

Puede ser una buena idea aplicar el concepto de oxidación en la reputación, esto es, cada cierto tiempo:

- Se suma a la reputación de los usuarios que están en puntos negativos algo de reputación.
- Se resta algo de reputación a los usuarios que tienen reputación positiva (a menos que se solicite una "baja" temporal).

La reputación buena debe oxidarse más rápidamente que la mala.

4.5. Trabajos relacionados

Al comienzo del proyecto realizamos un estudio de los proyectos existentes que tuviesen objetivos comunes a este. De ese estudio fue posible sacar las siguientes conclusiones:

- **Bancos de tiempo existentes.** En Internet existen una gran cantidad de bancos de tiempo como hOurWorld [55], TimeBanks [114] o incluso más cercanos como Ayudarnos.org [11] que a pesar de compartir la misma idea de banco de tiempo, están limitados por la necesidad de una administración central que se encargue de gestionar el sistema.

- **Bancos de tiempo P2P existentes.** En Internet existe una propuesta de banco de tiempo P2P [91] usando el sistema CHEs-ROCS [22], sin embargo no se ha encontrado ni especificada ni implementada. Aún así lo interesante de esta propuesta ha sido que el sistema CHEs-ROCS usa Ripple como protocolo de pago.

- **Protocolos de pago P2P.** En la actualidad el sistema P2P más utilizado para realizar pagos es Bitcoin [15]. Este sistema tiene un protocolo de pago que usa técnicas criptográficas para validar las transacciones [92].

  Otro protocolo de pago es Ripple [96]. Ripple consta de dos subsistemas complementarios. El primero es una versión moderna del antiguo sistema de Hawala [133] (de ahí el nombre Ripple"), en el que los usuarios tienen que especificar en quienes confían y hasta qué cantidad están dispuesto a avalar. El aval se puede especificar por activo. El segundo utiliza la moneda interna XRP. Para la seguridad de esta moneda, no se utiliza un sistema de minería como Bitcoin sino un algoritmo de consenso [56]. En 2013, Ripple Labs Inc. lanzaron el código fuente para los nodos P2P de Ripple en código abierto. Podría ser interesante para este proyecto estudiar en detalle el código fuente de Ripple pero no se hizo por falta de tiempo.

4.5.1. Otros trabajos relacionados

Existen otros posibles trabajos relacionados, que se mencionan a countinuación.

- Plugin banco de tiempo de Wordpress [144].
- Modulos para divisas sociales de Drupal (español) [59].
- Software para divisas de communidad (con participación de los más grandes en este campo) [88].
- PFC llamado "Gestión Web para un Banco del Tiempo" del año 2011 [33].

La conclusión del capítulo es que, a día de hoy, no hay ninguna tecnología P2P madura con la que se puede implementar un banco de tiempo fácilmente. Tampoco hay ninguna tecnología madura, robusta y mantenida con la que se puede simular un baco de tiempo; para un buen resultado, habría que hacerlo desde (prácticamente) cero sobre ns2 [76] o Simgrid [99]. Por tanto, en este proyecto, vamos a especificar bien el servicio que queremos e implementar solo una parte de ello.

Capítulo 5

Funcionalidad y requisitos

Al principio del proyecto se realizaron varias reuniones para establecer los posibles requisitos que debería tener una aplicación de banco de tiempo P2P. Además, en las primeras semanas se dedicó bastante tiempo para definir la funcionalidad del sistema. Todo ello se documentó usando de base algunos puntos de la especificación de requisitos software (IEEE Std 830-1998) [102].

5.1. Definición del sistema

- **Actores del sistema.** Al tener como objetivo la descentralización del sistema en una estructura de pares de nodos iguales, el sistema carece de cualquier tipo de administración y por lo tanto sólo se considera un único actor común en el sistema: el usuario.

- **Descripción del sistema.** Cada usuario debe registrarse previamente en el sistema para poder acceder al mismo. Una vez que ha accedido al sistema, el usuario puede ofertar o demandar servicios, y en caso de hacer lo segundo, tras la realización del servicio, deberá pagar con sus horas al usuario que ofertó el servicio la cantidad exigida en el acuerdo. Tanto el usuario acreedor como el usuario deudor podrán dar su opinión sobre el servicio realizado.

  A lo anterior se suma la posibilidad de los usuarios para realizar o solicitar préstamos e incluso la opción de ver un histórico de sus transacciones.

Por otro lado los servicios se catalogan de forma dinámica usando una ontología, pudiéndose establecer nuevas categorías más específicas.

- **Módulos del sistema.**
  - **Subsistema básico o de gestión de usuarios.**
    - **Gestión de usuario.** Módulo que permite al usuario registrarse en el sistema, iniciar y cerrar sesión, editar sus datos de perfil y desactivar la cuenta.
  - **Subsistema de transacciones.**
    - **Gestión de facturas.** Módulo que permite gestionar las transacciones del sistema. Es el núcleo bajo el cual los usuarios interaccionan y realizan el intercambio de servicios y tiempo. Pueden enviar presupuestos, facturas proforma, facturas y realizar el pago del servicio.
    - **Gestión de reputación.** Módulo que permite al usuario escribir, ver y eliminar comentarios y valoraciones sobre un servicio realizado/recibido.
    - **Gestión de balance/histórico.** Módulo que permite al usuario ver el balance de su cuenta.
    - **Gestión de préstamos.** Módulo que permite al usuario solicitar, hacer y ver préstamos.
  - **Subsistema de servicios.**
    - **Gestión de servicios.** Módulo que permite al usuario ofertar buscar, solicitar y cancelar un servicio.
    - **Gestión de ontología.** Módulo que permite crear una categoría de servicio en el sistema.
  - **Subsistema de notificaciones.**
    - **Gestión de notificaciones.** Módulo que permite al usuario enviar y recibir notificaciones.

5.2. Especificación del sistema

La especificación del sistema se agrupa en subsistemas. En cada subsistema aparecen las funciones descritas de forma objetiva. Cada función que

aparece en esta especificación se relaciona directamente con un caso de uso concreto en los documentos de análisis que se presentan más adelante en esta memoria. Para ver la especificación de los requisitos, ver Apéndice A.

Capítulo 6

Planificación, control y seguimiento

6.1. Planificación temporal

Para que todas las actividades se realizaran de manera eficiente y efectiva, al principio se optó por apuntar en un documento compartido las tareas asignadas a cada persona. La situación se empezó a hacer insostenible a mitad de proyecto, fue entonces cuando se optó por planificar sobre la herramienta de gestión de proyectos Redmine [95].

6.2. Metodología

Se trató desde el inicio erróneamente como una decisión rápida en el proyecto. En un principio se propuso realizar el desarrollo del proyecto siguiendo una metodología ágil, ya que el equipo estaba formado por tres miembros y había necesidad de ir viendo resultados parciales mientras se experimentaba con la tecnología P2P.

Entre las metodologías ágiles propuestas estaban Scrum y XP (eXtreme Programming):

6.2.1. Scrum

Scrum es una metodología ágil que busca realizar una serie de trabajos en equipo de manera eficiente, regular y colaborativa siguiendo los principios del Manifiesto Ágil [90] [140].

Scrum funciona en base a los Sprints (conjunto de tareas a realizar en un plazo de un mes). Cada Sprint consta de varias iteraciones. Cada iteración pasa por todas las fases del desarrollo (análisis, diseño, implementación, pruebas, puesta a producción, ...) y se realiza en un plazo de una semana como máximo [93].

6.2.2. XP (eXtreme Programming)

XP es una metodología ágil formulada por Kent Beck en 1999. El objetivo de esta metodología es intentar adaptarse rápidamente a los cambios en los requisitos software más que a preveer los posibles problemas a largo plazo. Por esta razón la documentación, las pruebas y los diseños quedan al margen [130].

Finalmente, como el proyecto era de naturaleza exploratoria, se optó por seguir eXtreme Programming de forma flexible como metodología concreta de desarrollo software ya que no se pretendía obtener un producto software concreto sino un prototipo que se pueda adaptar a distintas plataformas realizando pocos cambios.

6.3. Gestión de la configuración

Para poder realizar avances en el proyecto fue necesario el uso de repositorios web para dar soporte tanto a la documentación como a las implementaciones. En un principio la organización que se eligió fue la siguiente:

- Para gestionar la documentación: **Google Drive [50].**
- Para gestionar la planificación y el seguimiento: **Google Drive [50].**
- Para gestionar el código: **Google Code [49].**

Sin embargo, a mitad de proyecto, desde la plataforma de Google Code se nos notificó de la futura desaparición del mismo. Además se hizo necesario dar con una herramienta capaz de organizar mejor las tareas de cada alumno y evaluar los tiempos de los que se disponía. Fue entonces cuando se realizó un proceso de un par de semanas para cambiar a otras plataformas, que se utilizarían hasta el final del proyecto:

- Para gestionar la documentación: **Google Drive [50] y Source Forge[103].**
- Para gestionar la planificación y el seguimiento: **Hosted RedMine [54].**
- Para gestionar el código: **Source Forge [103].**

6.4. Líneas base

Puede considerarse como uno de los problemas al que nos enfrentamos en el desarrollo del proyecto. El hecho de no estructurar sobre herramientas sólidas desde el principio, tanto la documentación como los diagramas y el código, impidió que fuese posible tener líneas base sobre las que apoyarse para realizar los avances en el proyecto. Fue necesario tener que replantear las bases muchas veces, para poder avanzar, por la inmadurez de la tecnología y la falta de documentación.

6.5. Herramientas de planificación y seguimiento

En la fase inicial del proyecto se utilizaron para este propósito documentos compartidos en Google Drive [50] por todos los integrantes. A medida que el proyecto fue avanzando y creciendo se hizo necesario plantear una planificación, un seguimiento y un control más exhaustivo.

En este punto se introdujo Redmine [95], una herramienta de gestión de proyectos que incluye sistemas de seguimiento y control gracias a la posibilidad de generar calendarios de actividades y diagramas de Gantt como el que

se muestra en la Figura 6.1.

Además, en los últimos meses de proyecto, la frecuencia de las reuniones con el director se incrementaron notablemente, por lo que para complementar a esta herramienta se comenzó a generar actas de reunión, como la que se muestra en la Figura 6.2. Era necesario marcar objetivos a corto plazo y gracias a estas actas era posible reflejar qué temas se trataban en las reuniones y, lo más importante, cuáles eran las próximas acciones a realizar.

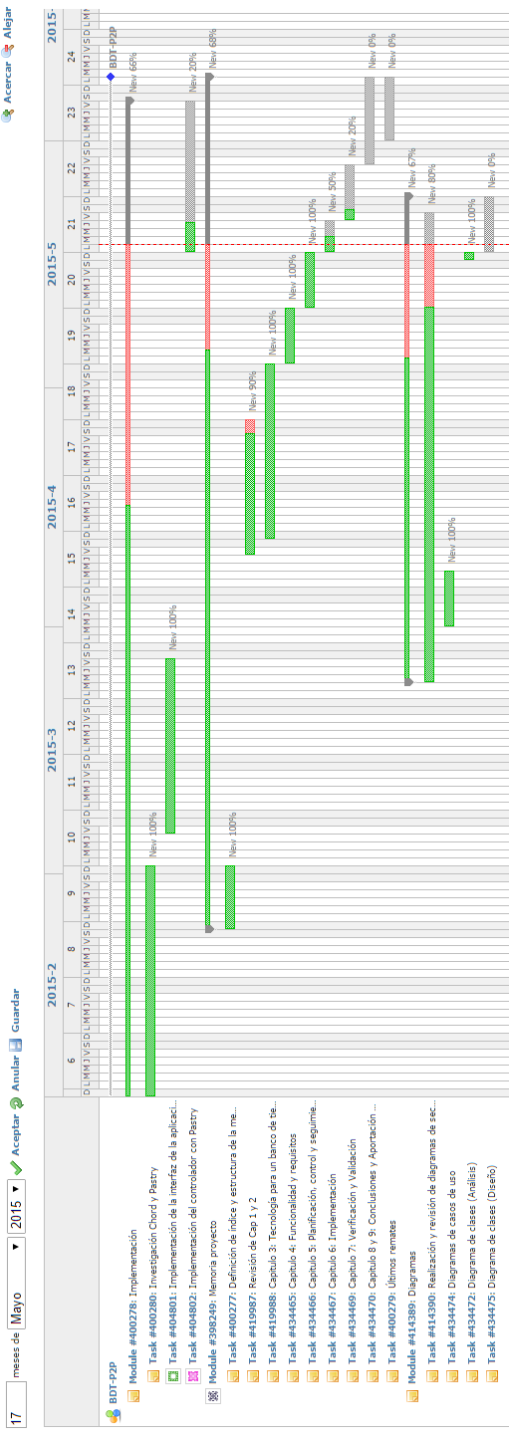

Figura 6.1: Diagrama de Gantt con Redmine.

Acta	[14/05/2015]		[UCM]

Reunión convocada por	Simon Pickin
Tipo de reunión	Informativa (Presencial)
Organizador	Simon Pickin
Apuntador	Daniel Alejandro Nowendsztern
Asistentes	Marcos Pérez García Antonio Núñez Guerrero Daniel Alejandro Nowendsztern Simon Pickin

Temas del orden del día

PLANIFICACIÓN

1. Debatir el sistema de ficheros propuesto por Simon.
2. Acordar los próximos trabajos a realizar

RESUMEN DE LA REUNIÓN

1. **Nuevo sistema de ficheros**:

Simon nos propuso un nuevo sistema de ficheros con las siguientes características principales:

- Ahora cada archivo únicamente tiene una entrada.
- El acceso es secuencial a partir del último fichero para acceder a los anteriores.
- Hay tres tipos de archivos (efímeros, temporales y permanentes), entendiendo efímeros los de 24h, temporales los de 1 mes y permanentes los de 5 años.
- Perfil privado (contraseña, lista de transacciones en curso, el UUID y clave privada) y perfil público para el resto.
- Aislar la parte mutable en un sitio muy pequeño, sólo se gestiona la mutabilidad del sistema en el profile.

2. **Future work:**
Simon nos propuso un auditor como software móvil para comprobar las transacciones (los caminos en el recorrido de los archivos de transacciones).

DISPOSICIÓN DEL TRABAJO

Tareas para Simon:

- **Hacer funciones base.**
- Hacer figura con las capas de los sistemas de ficheros (PAST, Chord...).
- Diagrama de pago (en diagrama de secuencia rudimentario, sólo intercambio de ficheros).
- Inicio de la semana que viene analizar la memoria.
- Estudiar protocolo de pago de BitCoin.

Tareas para el resto del equipo:

- Avanzar la memoria.
- Revisar el nuevo sistema de ficheros.
- Hacer diagrama de análisis, usar JSON para sistema de ficheros.

Figura 6.2: Acta de reunión.

Capítulo 7

Implementación

7.1. Arquitectura

La arquitectura del prototipo engloba tanto los patrones de diseño como las plataformas utilizadas para el desarrollo del mismo. Además de los diagramas que componen la especificación.

7.1.1. Patrones de diseño

En esta sección se explicarán algunos de los patrones de diseño más importantes que se han aplicado en el desarrollo del prototipo de la aplicación.

MVC

El patrón modelo de vista del controlador (MVC), separa presentación e interacción de los datos del sistema.

El componente Modelo maneja los datos del sistema y las operaciones asociadas a esos datos.

El componente Vista define y gestiona cómo se presentan los datos al usuario.

El componente Controlador dirige la interacción del usuario (por ejemplo, teclas oprimidas, clics del mouse, etcétera) y pasa estas interacciones a Vista y Modelo [57].

En nuestro caso, el modelo se corresponde con el paquete *model* que representan al sistema de ficheros que aparece en el Apéndice D. Las vistas con el paquete *gui*, que presentan los datos al usuario y envían las peticiones del usuario a la parte del controlador que se encarga de gestionar los eventos de la vista. El controlador con el paquete *controller* que tiene una estructura que busca ser adaptable a distintas plataformas P2P y tiene la obligación de gestionar las conexiones a la red, el almacenamiento de los ficheros (funciones para poner y obtener ficheros de la red P2P) y las operaciones propias del sistema (registrar usuario, iniciar sesión, efectuar pago, etc. . .).

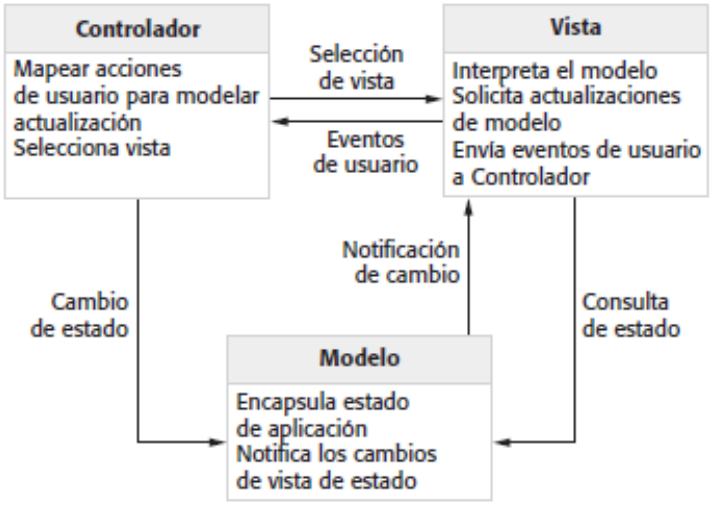

Figura 7.1: Diagrama del MVC [57].

Singleton

El patrón *Singleton* se asegura de que exista una única instancia de una clase y provee un punto de acceso global a dicha instancia [31].

En nuestro caso, se usó el patrón *Singleton* en la clase *Util* encargada de realizar operaciones generales que pudieran ser útiles para el resto del sistema.

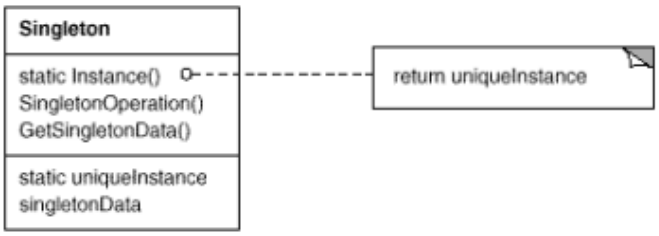

Figura 7.2: Diagrama del patrón Singleton[31].

Fachada

El patrón fachada o *façade* provee una única interfaz de comunicación con un sistema o subsistema complejo [31].

En nuestro caso, se utilizó el patrón fachada en la interfaz *IManager* encargada de proveer un acceso único a las operaciones de todos los subsistemas (servicios, transacciones, etc. . .).

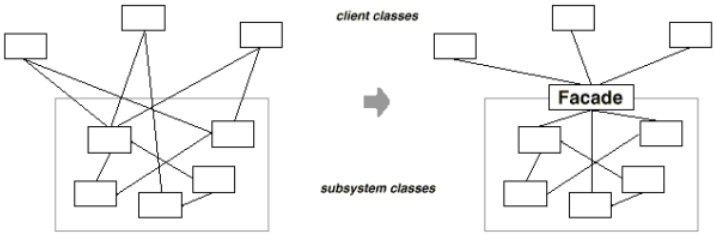

Figura 7.3: Diagrama del patrón Fachada[31].

Otro patrón de diseño útil para la implementación hubiera sido el patrón **"cadena de responsabilidad"** (en inglés *chain of responsibility*) cuyo objetivo es realizar una serie de operaciones en un orden determinado (aún siendo operaciones asíncronas). En nuestro caso hubiera sido interesante usar este patrón para evitar un anidamiento excesivo de *callbacks*.

7.1.2. Decisiones de diseño

En esta sección se comentarán las decisiones de diseño más importantes que se han tomado.

- **División de la información en múltiples archivos.** Pensando en la estructura de una DHT y en como modificar y borrar la información que se almacena en ella se decidió dividir las múltiples entradas de cada archivo en distintos ficheros separados y enlazados en la red. La ventaja principal que se obtiene de esta disposición es la posibilidad de que entradas concretas expiren con el paso del tiempo, eliminándose de la red. La modificación y el borrado se convierte en *garbage collection* [131]. La principal desventaja es la seguridad, puesto que es dificil asegurar una secuencia de transacciones, ante los problemas de numeración si se empiezan a llevar a cabo varias transacciones sucesoras simultáneamente. Para intentar reducir este problema de seguridad, la idea es que las transacciones sean firmadas por ambas partes (deudor y acreedor).

- **Idea de comentarios escritos por mí hacia otros y hechos por otros hacia mí.** Esta idea surge para complementar al sistema de reputación automático y tiene como ventaja aumentar la confianza entre los

usuarios del sistema. Tiene como desventaja el aumento de recursos de almacenamiento y su gestión en la red.

- **Facturas y presupuestos.** Se diseñó pensando en hechos de la vida real como puede ser en el sector de la construcción donde el proceso requiere un presupuesto, una factura proforma y una factura.
- **Firma digital.** Por razones de seguridad e integridad de datos, cada fichero del sistema tiene un propietario quien firma digitalmente su contenido.

7.1.3. Plataformas usadas

Durante la implementación del prototipo se han realizado diversas pruebas con distintas implementaciones de Chord (un protocolo P2P de búsqueda distribuída que asocia claves con nodos [126]), el objetivo de las pruebas era que el sistema funcionase correctamente tanto en local como en Internet. Además se buscaba que ofreciese la posibilidad de hacer simulaciones sobre la propia implementación.

Todas las implementaciones de Chord que se probaron no dieron resultados positivos a estas pruebas salvo uno (Open Chord [77]), por esta razón fue necesario probar con otra tecnología P2P, es decir, se realizaron pruebas sobre una implementación de Pastry. El resultado de las pruebas en una implementación de Pastry [138] (FreePastry [44]) mejoró al resultado del mejor framework de Chord. Luego se optó por empezar a implementar el prototipo a partir de esta tecnología.

Aunque más tarde, cuando el proyecto estaba más avanzado se descubrió otro tecnología similar que estaba implementada (Kademlia [135]), pero que no se había probado.

Open Chord

Open Chord es una implementación de código abierto de Chord. Está disponible de forma gratuita bajo licencia GNU GPL y ha sido desarrollado por el Grupo de Sistemas Distribuidos móvil y de la Universidad de Bamberg.

De la misma manera que una base de datos orientada a objetos, Open Chord ofrece la posibilidad de utilizar la tabla hash distribuída dentro de las aplicaciones Java, proporcionando una API para almacenar todos los objetos Java serializables dentro de la tabla hash distribuida [77].

FreePastry

FreePastry es un framework genérico, escalable y eficiente para aplicaciones P2P. Está formada por nodos, en una red superpuesta bajo una autoorganización descentralizada y con tolerancia a fallos de la red.

FreePastry también proporciona enrutamiento eficaz, localización determinista de objetos y equilibrio de carga de manera independiente a la aplicación. Además, FreePastry proporciona mecanismos que apoyan y facilitan la replicación de los datos, el almacenamiento en caché y la recuperación en caso de fallo [44].

En el tema de la documentación, FreePastry es más completo con respecto a otras implementaciones, ya que incluye código comentado, ejemplos y tutoriales en su página web, aunque todavía tiene mucho camino por recorrer.

Figura 7.4: Logotipo de FreePastry. [44].

7.1.4. Ficheros del sistema

Los ficheros del sistema se consideran como una pieza fundamental del proyecto. Para cada fichero se define la mutabilidad, la persistencia y la seguridad del mismo. El conjunto de ficheros del sistema se puede ver en el Apéndice D

7.1.5. Funciones primitivas

La funciones primitivas se idearon en este proyecto como una manera de unificar las acciones a realizar para obtener y almacenar el contenido de los ficheros en la tabla hash distribuída de una plataforma P2P. Para más detalles, consultar el Apéndice E

7.1.6. Diagramas

Para la representación del sistema se realizaron diagramas UML 2.x. Los diagramas más importantes del sistema, es decir, los que se consideran imprecindibles para entender el sistema o los que no son triviales. Los diagramas de análisis se muestran en Apéndice B. Los diagramas de diseño se muestran en el Apéndice C.1.

7.2. Interfaces gráficas

A la hora de desarrollar las vistas de la interfaz gráfica de la aplicación nos hemos basado en las que ofrece TimeOverflow [115], un software diseñado por y para los banco de tiempo, en su repositorio de GitHub [116].

En el breve periodo de implementación realizamos unas maquetas con algunas de las interfaces gráficas que se deseaban mostrar en el prototipo:

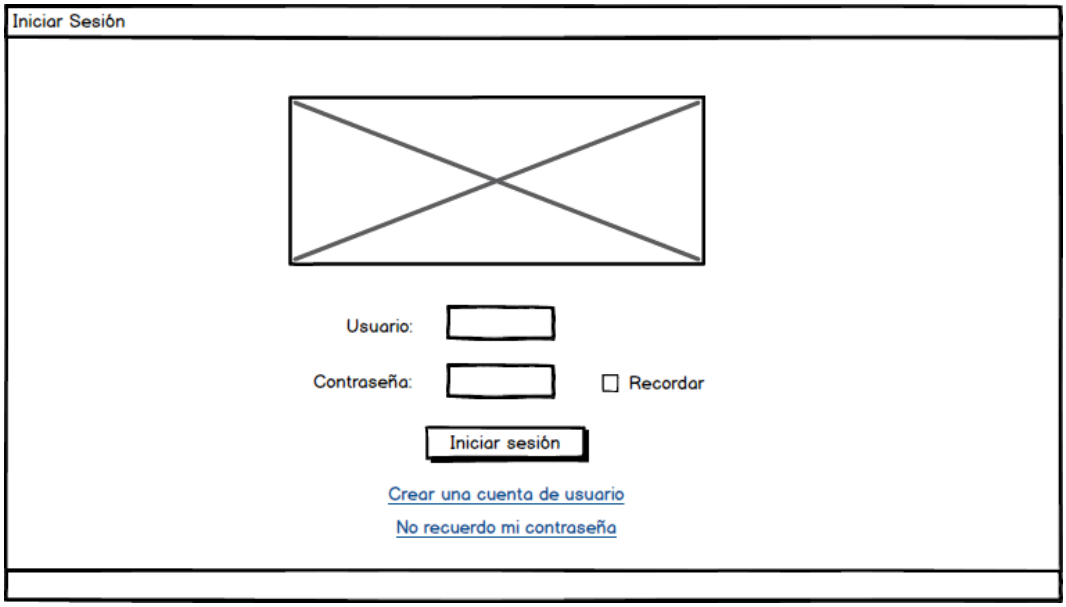

Figura 7.5: Ventana de iniciar sesión.

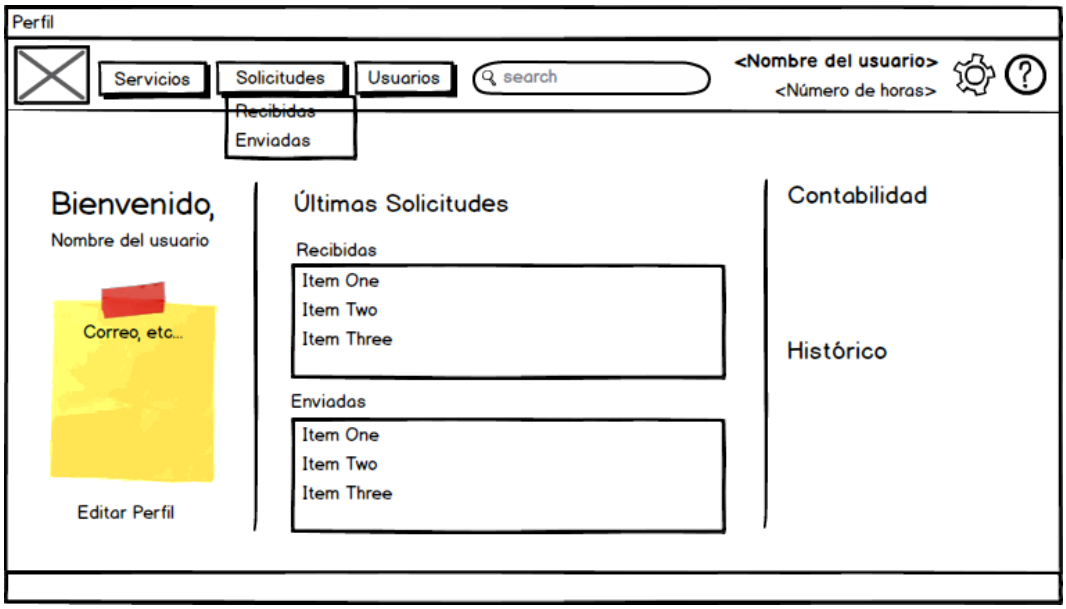

Figura 7.6: Ventana de perfil.

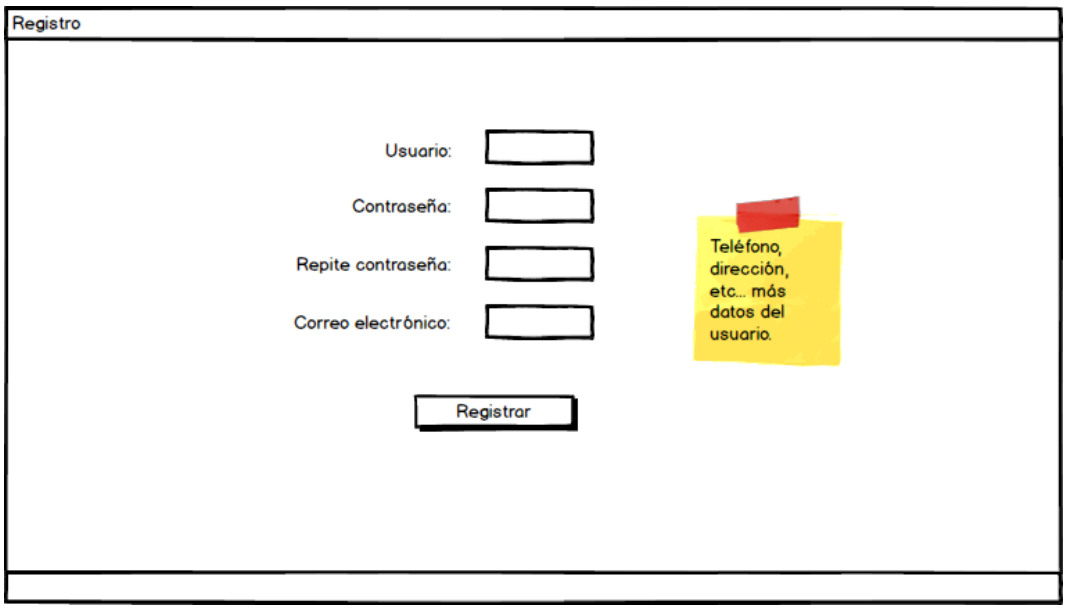

Figura 7.7: Ventana de registro.

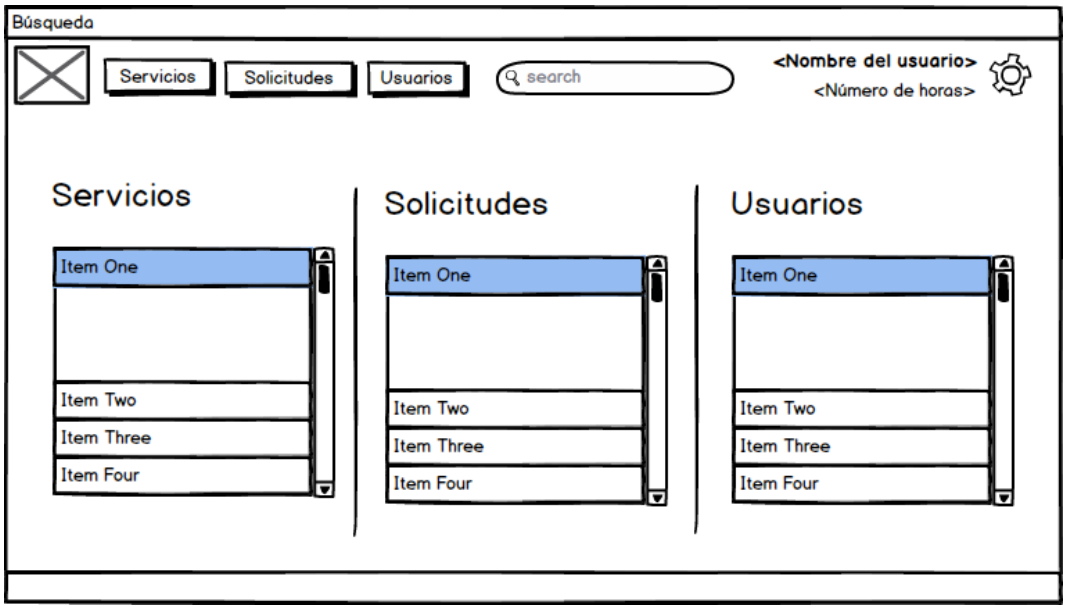

Figura 7.8: Ventana de búsqueda.

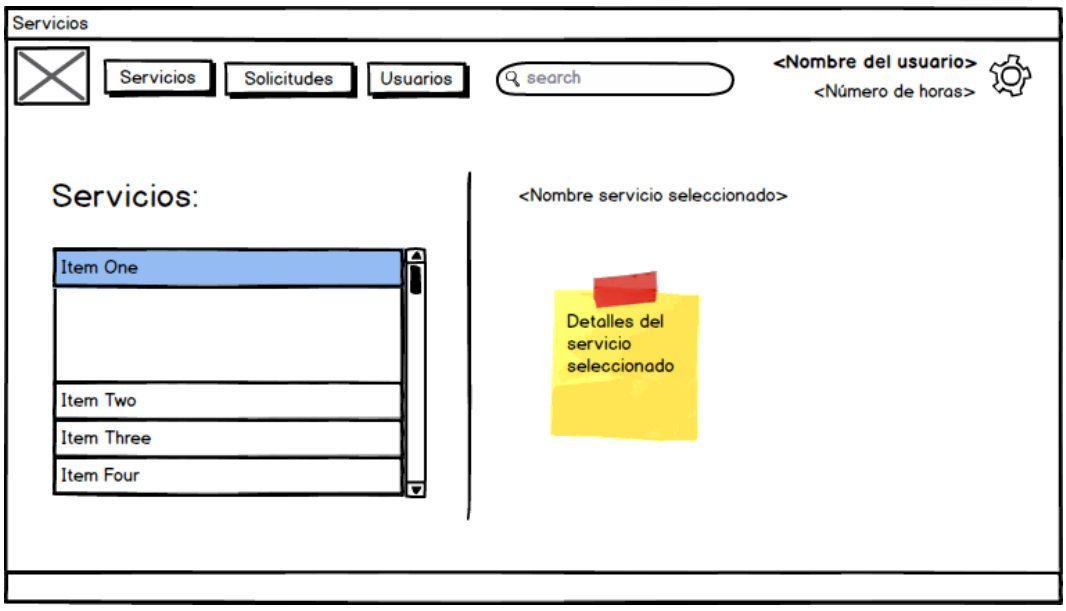

Figura 7.9: Ventana de servicios.

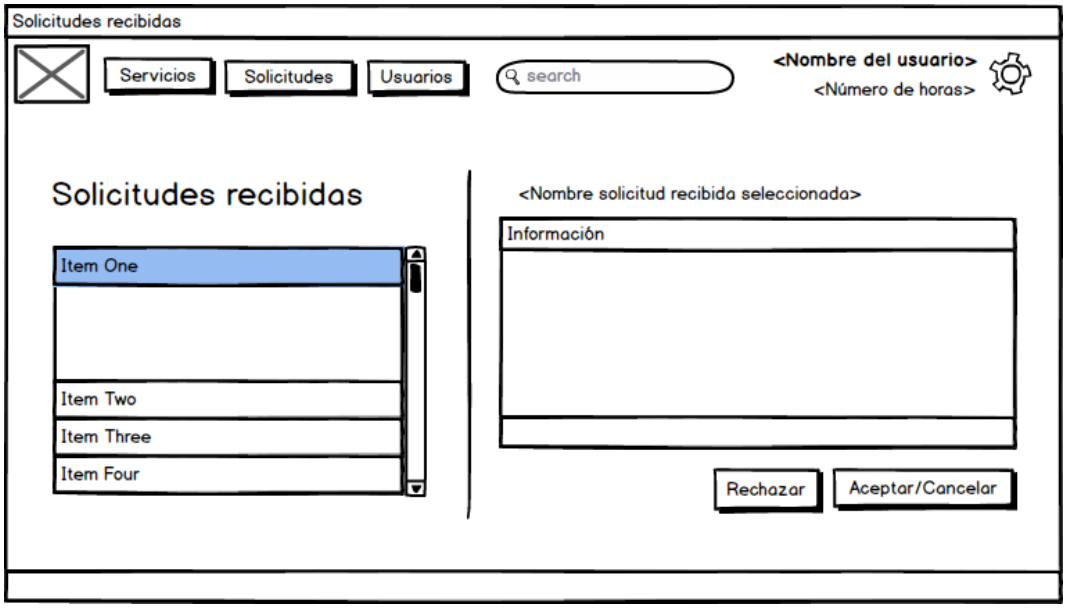

Figura 7.10: Ventana de solicitudes recibidas.

7.3. Herramientas

En esta sección se describirán las herramientas utilizadas en el desarrollo del prototipo.

7.3.1. Herramientas de modelado

A lo largo del desarrollo se han utilizado una gran cantidad de herramientas de modelado. En un principio se decidió utilizar la herramienta Modelio [72] en su versión gratuita de código abierto, pero tenía un inconveniente importante: la imposibilidad para trabajar en equipo bajo algún repositorio. Además la aplicación impedía copiar y pegar entre diagramas, aunque se entiende que esta imposibilidad es la consecuencia de obligar al usuario a trabajar al nivel de modelo.

De todas formas, la decisión fue rápida, era necesario buscar otra herramienta. La que más se recomendaba en la red era Visual Paradigm 12 [120] en versión comunitaria, que tras ser probado con diagramas de secuencia parecía ser la mejor opción. Pero no lo fue, porque aunque se podía trabajar en paralelo bajo un repositorio e incluso realizar las acciones de copiar y pegar en diagramas, ya cuando los diagramas estaban realizados no era posible sacar capturas de pantalla sin marcas de agua que le quitaran legibilidad.

El tema se complicaba poco a poco, ya iban dos decisiones equivocadas, pero aún había esperanzas de dar con la herramienta deseada. Fue entonces cuando se consiguió la herramienta Enterprise Architect 8 [42], que después de muchas pruebas paso a ser la solución perfecta. Sin embargo no lo era, pues tenía tres problemas invisibles a primera vista, el primero era la falta de compatibilidad entre versiones de distintos sistemas operativos, el segundo era las dificultades para realizar modificaciones sobre los diagramas ya existentes y el tercero era la limitación de tiempo de la licencia (30 días).

El tiempo del proyecto se agotaba. Para la entrega faltaba menos de tres meses. Ya no quedaban muchas más opciones. En un principio se decidió por la herramienta Eclipse IDE[40] que con el plugin Papyrus[82] solventaba muchos de los problemas del resto de herramientas. Sin embargo esta herramienta por muy completa que fuera dificultaba mucho la modificación de los

diagramas realizados, por ser herramienta de modelado, no de dibujo.

Finalmente, ante la falta de tiempo, se decidió usar la herramienta de dibujo draw.io [36], que se ajustaba a nuestras necesidades. Esta herramienta ofrece las posibilidades de una herramienta de edición completa (opciones de copiar, pegar, deshacer, rehacer, etcétera), con la capacidad de edición online y además está soportada por la herramienta de colaboración en tiempo real conocida actualmente como Google Drive [50].

7.3.2. Herramientas de implementación

La implementación del prototipo se ha realizado con Eclipse IDE. Eclipse IDE [40] es una plataforma de código abierto basada en Java [60] que permite a los desarrolladores de software crear un entorno de desarrollo integrado (IDE) con componentes construidos por miembros de Eclipse.

En particular, se ha utilizado una versión de Eclipse IDE conocida como e(fx)clipse [41], que además provee las herramientas necesarias para desarrollar bajo la tecnología JavaFX [134].

7.3.3. Herramientas de documentación

Para realizar la documentación se decidió usar el sistema de composición de textos LaTeX [65] porque facilita el trabajo en equipo mediante el almacenamiento en algún repositorio y además dota al texto de un aspecto más profesional. En un principio se pensó en usar programas como TexStudio [109] o LatexEditor [66], pero ninguno funcionaba correctamente con repositorios en Internet. Por esa razón se decidió usar un plugin de Eclipse IDE. El plugin se llama Texlipse [108].

7.3.4. Herramientas de comunicación

Para la comunicación se utilizaron las siguientes herramientas software: WhatsApp [122], Teamviewer [107] y Skype [100].

Capítulo 8

Verificación y Validación

En este capítulo se explicarán los métodos que se podrían utilizar para verificar (comprobar que el software está de acuerdo con su especificación) y validar (comprobar que el software cumple las expectativas del cliente) las aplicaciones P2P y, en particular, nuestro prototipo de aplicación [57], indicando asimismo cuales hemos utilizado.

8.1. Simulación

No fue posible realizar ningún tipo de simulación del prototipo debido a la falta de simuladores consolidados para esta tecnología, y a la falta de tiempo en el caso de FreePastry's Simulator, que fue el candidato principal para la simulación.

8.1.1. Posibles herramientas de simulación.

Las reseñas de algunos de los simuladores que fueron encontrados en Internet se muestran a continuación.

- **Nombre:** FreePastry's Simulator [45]
  Lenguaje de programación: Java
  Última versión conocida: 2012
  Estado: Buena documentación e interesante implementación.
- **Nombre:** OverSim [79]
  Lenguaje de programación: C

Última versión conocida: 2012
Estado: Buena documentación.

- **Nombre:** PeerSim [86]
  Lenguaje de programación: Java
  Última versión conocida: 2009
  Estado: No está bien documentado.

- **Nombre:** SimGrid [99]
  Lenguaje de programación: Java
  Última versión conocida: 06/2014
  Estado: No tiene implementación P2P.

- **Nombre:** P2PSim [81]
  Lenguaje de programación: C
  Última versión conocida: 2005
  Estado: No está bien documentado.

- **Nombre:** PeerfactSim.KOM [85]
  Lenguaje de programación: Java
  Última versión conocida: 2011
  Estado: Buena documentación y buena implementación.

- **Nombre:** P2P Trust Simulator [80]
  Lenguaje de programación: Java y C
  Última versión conocida: 25/10/2009
  Estado: Buena documentación.

- **Nombre:** d-p2p-sim [27]
  Lenguaje de programación: Java
  Última versión conocida: 2010
  Estado: Escasa documentación.

- **Nombre:** PeerThing [87]
  Lenguaje de programación: Java
  Última versión conocida: 2013
  Estado: Falta documentación.

8.1.2. Simulación sobre FreePastry's Simulator

FreePastry's Simulator [45] es un simulador discreto de eventos. Es capaz de ejecutar aplicaciones FreePastry sin modificaciones en el código fuente. Ejecuta el código más rápido que en tiempo real aunque también puede ser ejecutado en tiempo del sistema. Además, acepta una variedad de topologías de latencia de red.

Simulación sobre OverSim

OverSim es un framework que actúa como una capa por encima de la implementación [79].

Cuenta con soporte para modelos de red P2P estructurados (Chord, Kademlia, Pastry, etc...) y no estructurados (GIA).

En el proyecto no fue posible probar el prototipo con este framework antes de la entrega, por falta de tiempo.

8.2. Pruebas

En la actualidad no se encuentra mucha información sobre pruebas de aplicaciones P2P.

En la finalización del proyecto se intentaron realizar algunas pruebas activas al prototipo implementado. Aunque sólo fue posible llevar a cabo un tipo de prueba, las pruebas unitarias de funcionalidad. Quizás hubiera sido muy interesante poder realizar otras pruebas tales como las de robustez y carga que se describen en esta sección.

8.2.1. Pruebas activas

Entre todos los tipos de pruebas activas, se consideraron como importantes para este proyecto los siguientes tres tipos de pruebas: las pruebas

unitarias de funcionalidad, las pruebas de robustez y las pruebas de carga.

Pruebas unitarias de funcionalidad

Estas pruebas estaban dirigidas a comprobar el funcionamiento de la parte implementada del prototipo. Se realizaron usando el framework JUnit [63]. Para ello se diseñaron una serie de funciones que se encargarían de comprobar la validez de los datos después de la ejecución de una operación. Todas las pruebas se pueden comprobar en la implementación que se adjunta en la entrega del proyecto.

Pruebas de robustez

Habría sido interesante poder comprobar la respuesta del sistema ante la caída de varios nodos de la red y la entrada de otros (*churn*). Además se podría haber realizado una prueba con muchos nodos en la cual se "sufre un apagón inesperado en la red" y sólo queda un nodo con la información del resto. A continuación se conectarían todos los restantes otra vez. Algunas preguntas que podrían surgir: ¿se mantendría toda la información del sistema en ese único nodo activo?, ¿cómo y en qué momento se distribuiría esa información con la entrada de los nodos?, y ¿podrían continuar las operaciones en el sistema mientras ocurren estas acciones?.

Pruebas de carga

También hubiera sido interesante comprobar la respuesta del sistema a la realización de una gran cantidad de transacciones en un mismo instante de tiempo desde un punto de acceso.

Pruebas de integración

Se entiende por pruebas de integración a las pruebas de subsistemas cada vez más grandes.

Este tipo de pruebas no se realizaron, debido a que no hay mucho sitio para definir subsistemas cada vez más grandes, porque los nodos son todos iguales, por tanto aquí es difícil encontrar varios niveles de subsistema (quizás, en un mismo nodo, la parte red, la parte comunicaciones, etc...). Aunque también se podría interpretar como pruebas de integración a probar configuraciones con funcionalidadades que van aumentando.

Por tanto, se podrían realizar las siguientes pruebas:

- Subsistemas nuestros.
- Subsistemas sin interfaz gráfica o con interfaz gráfica.
- Configuraciones con aumento de funcionalidad.
    - Pocos nodos, sin *churn*
    - Pocos nodos, con *churn*
    - Muchos nodos, sin *churn*
    - Muchos nodos, con *churn*

Más información sobre las pruebas de configuraciones con aumento de funcionalidad en [2].

8.2.2. Pruebas pasivas

Solo tiene sentido hablar de pruebas pasivas cuando se trata de pruebas de un sistema. ¿Pero qué es el sistema?, un código que puede ejecutarse sin un *"test harness"*, es decir, solo con usuarios externos. El prototipo de un banco de tiempo sin la parte ofertas-demandas y sin sistema de reputación automático, todavía puede considerarse un sistema, pero la interfaz gráfica de un nodo, por ejemplo, no puede.

Prueba pasiva del banco de tiempo

En nuestro caso, la prueba pasiva / monitorización puede ser muy interesante. Un ejemplo sería, a partir de la última transacción de un usuario, seguir los enlaces hacia transacciones pasadas para comprobar que no hay enlaces defectuosos o ficheros que se han perdido. También se podría rehacer

los chequeos que hacen los nodos en el curso de una transacción. Se podría incluso concebir código móvil auditor, pero en ese caso surgiría la cuestión de cómo se sabe que el probador no es código malicioso.

8.3. Banco de pruebas (*testbed*)

Un banco de pruebas es una plataforma para experimentación de proyectos de gran desarrollo [125]. A nivel europeo existe un banco de pruebas en el que participan diversas universidades de Europa, dicho banco de pruebas se conoce como PlanetLab [89]. También existen otros como es el caso del Community Networks Testbed [23]. Sería interesante poder ejecutar nuestra aplicación en un *testbed* de este tipo.

Capítulo 9

Conclusiones

Se ha realizado un proyecto muy complejo en el que, en un corto periodo de tiempo, ha sido necesario enfrentarse a diversas tecnologías P2P no muy conocidas y construidas en unas infraestructuras P2P no muy fiables y poco documentadas.

El primer objetivo ha sido el estudio y aprendizaje de la tecnología P2P, además de documentar el conocimiento adquirido en este campo en la descripción del estado del arte, antes de crear la especificación informal del sistema. Por culpa de la relativa inmadurez de la tecnología P2P, después de estudiar el estado del arte en esta área, fue difícil pasar directamente a la definición de los requisitos del sistema sin primero esclarecer cómo un sistema podría construirse usando la tecnología actual. Por esta razón, y por la insistencia de nuestro director del proyecto, la especificación informal es bastante completa. Este objetivo se ha logrado con la dificultad de ser un campo nebuloso que no es obvio, como se puede ver en el capítulo 3 del estado del arte y en el capítulo 4 con la especificación informal.

El segundo objetivo era producir una especificación más formal del sistema compuesta por:

- **Análisis**
    - Especificación de requisitos. Necesario para establecer las bases del proyecto. Ver Apéndice A.
    - Diagramas de casos de uso. Estos diagramas fueron elegidos puesto

que daban una visión global de la funcionalidad del sistema. Ver la sección B.1.

- Diagrama de clases. Creados en la fase de análisis para definir el modelo de dominio, que es la representación estructural de los conceptos principales del problema, y que contienen también la vista estructural de alto nivel del software que se deseaba construir. En nuestro caso, el modelo de dominio era suficiente como propuesta para entender mejor los requisitos y esclarecer los conceptos principales del problema de dominio. Ver sección B.2.
- Diagramas de secuencia. Los diagramas de secuencia se eligieron porque eran más adecuados que otros diagramas, como los de actividad, para un sistema distribuido basado en intercambio de mensajes. Ver sección B.3.
- Sistema de ficheros. Pieza fundamental de la parte del sistema especificado, que de hecho, es un tipo de base de datos distribuída. Las "tablas" que se corresponden con el esquema de la base de datos son los ficheros del sistema, y están en el Apéndice D.

Por la insistencia de nuestro director de proyecto, invertimos una gran cantidad de esfuerzo en asegurar que la especificación del análisis del sistema fuera de buena calidad, en particular, los diagramas de secuencia.

- **Diseño**

  - Diagrama de clases. Usado en la fase de diseño para describir en detalle el diseño de la parte del sistema especificada que se debía implementar en el prototipo. Ver C.1.

Este objetivo se ha cumplido parcialmente ya que aunque se ha realizado la especificación completa del subsistema de gestión de usuarios y del módulo de gestión de facturas, se han realizado parcialmente los módulos de gestión de reputación, de gestión de balance/histórico, de gestión de préstamos, de gestión de servicios, el de gestión de notificaciones y no se ha realizado el de gestión de ontología.

Otro de los objetivos había sido la implementación de una parte del prototipo. Hasta la fecha se ha conseguido implementar el subsistema básico de ges-

tión de usuarios y la estructura para el módulo de gestión de facturas. Se puede considerar que se ha cumplido parcialmente. Para más información, consultar el proyecto en el repositorio: *http://sourceforge.net/p/bancodetiempoucm/svn/HEAD/tree/*

Finalmente, otro de los objetivos era realizar una verificación y validación de nuestro sistema. Como no hemos logrado ni simular ni probar nuestra aplicación, el objetivo no se ha logrado. De todas formas, una de las consecuencias de la inmadurez del campo de la tecnología P2P es la dificultad para producir una especificación del sistema sin primero esclarecer como ese sistema podría construirse usando la tecnología actualmente disponible, por tanto también es difícil verficar y validar nuestra aplicación sin primero esclarecer qué tipo de V&V es posible. Un estudio de este tipo de V&V se podría usar en aplicaciones P2P hechas y descritas en el capítulo 8.

A lo largo del proyecto se han identificado diversos problemas entre los cuales están:

- Problemas a la hora de encontrar plataformas P2P estables para realizar todas las actividades del proyecto.
- Algunos errores en la planificación y el seguimiento, sobre todo al inicio.
- Dificultades para avanzar por falta de conocimientos sobre la materia.

9.1. Trabajo futuro

Como propuesta a futuro, sería interesante realizar las siguientes actividades:

- Especificar las partes del sistema que quedaron sin especificar o que no se especificaron por completo, como el subsistema de gestión de servicios, un sistema de reputación más sofisticado y el de gestión de prestamos.
- Implementar las partes del sistema que quedaron sin implementar o que no se implementaron por completo. En el caso del sistema de reputación, simulación del sistema o ejecución de *test-bed* es necesario comprobar cuales de ellos son viables y realistas.

- Estudiar la posibilidad de integrar mecanismo de certificación de identidad de tipo "web de confianza" de PGP como opción extra para los usuarios que quieran.
- Simular el prototipo sobre OverSim o FreePastry's Simulator.
- Incorporar relaciones de uno a muchos o de muchos a muchos en las transacciones.
- Estudiar el código *open source* del protocolo de pago Ripple.
- Explorar la posibilidad de mejorar más la verificación y validación de nuestra aplicación.
- Agregar a la implementación las propuestas para el sistema de reputación con el objetivo de saber si estas propuestas son factibles / realistas.
- Completar el tratamiento de errores, en particular el aborto / *rollback* de transacciones si una de la dos partes no contesta.
- Comparar nuestro modelo de datos con el de Community Weaver [24] y hOurWorld [55].

Capítulo 10

Conclusions

We have carried out a very complex project in which, in a short period of time, we have had to deal with a range of not very well-known P2P technologies and build on not very reliable and poorly documented P2P infrastructures.

The first objective was to study and understand P2P technology and to capture the knowledge acquired in a description of the state of the art in this field, before going on to create an informal specification of the system. Due to the relative immaturity of P2P technology, after studying the state of the art in this area, it was difficult to proceed directly to defining the required system without first clarifying how such a system could be built using currently-available technology. For this reason, and at the insistence of our project supervisor, the informal specification is quite comprehensive. This objective was achieved, in a knowledge area that is currently rather nebulous and non-obvious, see Chapter 3 for the state of the art, and Chapter 4 for the informal specification.

The second objective was to produce a more formal specification of the required system composed of:

- **Analysis**
    - Requirements specification. Needed to establish the basis of the project. See Appendix A.
    - Use-case diagrams. Used to give an overview of the system funcionaity. See Section B.1

- Class diagrams. Used in the analysis phase to define the domain model, that is a structural representation of the main concepts of the problem domain, though they may also contain a high-level structural view of the software system to be built. In our case, the domain model was sufficient for the purpose of increasing understanding of the requirements and clarifying the main concepts of the problem domain. See Section B.2.
- Sequence diagrams. Used in the analysis phase to define usage scenarios, that is, high-level behavioural views of the use-case implementations. Sequence diagrams were chosen instead of, say, activity diagrams since UML2 sequence diagrams are the most suitable diagrams for specifying distributed system behaviour based on message exchange. See Section B.3.
- Data model. A fundamental part of the system being specified is, in effect, a type of distributed database. The "tables" of the corresponding database schema are the system files defined in Appendix D.

At the insistence of our project supervisor, a considerable amount of effort went into ensuring that the specifications of our system analysis were of good quality, in particular, the sequence diagrams.

- **Design**
  - Class diagrams. Used in the design phase to describe the detailed design of the part of the system to be implemented in the prototype. See C.1.

This objective has been partially achieved since, although a complete specification of the user-management and transaction-management subsystems has been defined, the specification of the reputation, balance/history management, loan management, service-management and notification modules has been only partially specified, and the ontology-management module has not been specified at all.

Another of the objectives was the implementation of a prototype. To date, we have implemented the user-management subsystem and the structure of the transaction-management subsystem, for which reason it can be considered to have been partially achieved. For more information, see the project

repository at *http://sourceforge.net/p/bancodetiempoucm/svn/HEAD/tree/*

Finally, another objective was to perform verification and validation on our system. Since we have managed neither to simulate nor to test our application, this objective has not been achieved. However, just as one consequence of the immaturity of the field of P2P technology is that it was difficult to produce the system specification without first clarifying how such a system could be built using currently-available technology, it would also be difficult to proceed to verifying and validating our application without first clarifying what type of V&V is possible. A study of the type of V&V that could be used with P2P applications was carried out and is reported on in Chapter 8.

In the course of the project the following problems arose:

- Problems finding stable P2P platforms on which to base all project activities.
- Some errors in the planning and monitoring, particularly at the beginning.
- Difficulties in moving forward due to lack of knowledge on the subject.

10.1. Future Work

As a suggestion for future work, it would be interesting to conduct the following activities:

- Specify the parts of the system that weren't specified or were incompletely specified, such as the service management, a more sophisticated reputation system, loans management, etc.
- Implement the parts of the system that weren't implemented or were incompletely implemented. In the case of the reputation system, simulation or test-bed execution is needed to check which of them are feasible and realistic.
- Study the possibility of incorporating a PGP web of trust type of certification mechanism as an optional extra for users who wish to include it.

- Simulate the prototype on OverSim or FreePastry's Simulator.
- Incorporate one-to-many or many-to-many transactions.
- Study the open-source code of the Ripple payment protocol.
- Explore the possibility of performing more verification and validation of our application.
- Complete the treatment of errors, in particular rollback of transactions in the case where one of the two parties does not answer.
- Compare the data model of our application with that of Community Weaver [24] and hOurWorld [55].

Capítulo 11

Aportación de cada alumno al proyecto

Este proyecto se compone de muchas actividades que debieron ser tratadas en común. La primera actividad en común que se realizó fue la de proponer el proyecto al profesor, bajo la única condición de que se desarrollara bajo la tecnología P2P. Después de haberlo definido, se empezó a trabajar en los requisitos y las funcionalidades del sistema. A continuación se llevó a cabo el modelado con diagramas de distintos tipos (casos de uso, secuencia y clases) que fueron revisados múltiples veces durante el ciclo del proyecto. Después se realizó un trabajo de investigación sobre las herramientas (plataformas, simuladores, herramientas de modelado, herramientas de desarrollo software y de documentación...). Finalmente se realizó la memoria y la implementación de una parte del prototipo.

A continuación se detallan algunos de los trabajos individuales que se realizaron en las actividades anteriormente mencionadas.

11.1. Antonio Núñez Guerrero

La primera aportación fue la idea inicial. Propuso esta idea porque, al ver que en la descripción del proyecto decía que se debía desarrollar una aplicación de utilidad social implementada bajo la tecnología P2P, se le ocurrió la idea de proponer como aplicación un banco de tiempo. La causa de la esta

decisión fue una clase de Ética Legislación y Profesión en la cual se habló de ésta idea y le pareció una idea bastante interesante para los tiempos que corría en España. Además, en el cuatrimestre siguiente, llevó a cabo la implementación de una página web de un banco de tiempo para la asignatura Aplicaciones Web. Cuando vió que la propuesta de proyecto reunía la condición de que dicha aplicación estuviera desarrollada en P2P, se le ocurrió que podría ser original la idea de llevar a cabo un banco de tiempo en P2P.

Una vez reunidos los alumnos con el profesor, les habló de que quizás se debería añadir a la funcionalidad de la aplicación un un sistema de reputación, aunque no fuera en este proyecto, sino en una continuación de éste. Aconsejó esto porque pretendía que la aplicación resultante consiguiera un mínimo de confianza para los usuarios y evitar la presencia de usuarios maliciosos en el sistema, dando lugar así a una comunidad participativa, servicial y más justa.

Como parte de la tarea del análisis de requisitos, se encargó de llevar a cabo un primer análisis de los bancos de tiempo actuales más relevantes y realizó una búsqueda de información sobre estos. Así consiguió averiguar requisitos necesarios importantes como la necesidad de fomentar las relaciones entre los usuarios y aspectos innecesarios en esta aplicación como la existencia de un administrador de sistema.

Una vez estudiados los requisitos, elaboró la definición de los módulos y las funcionalidades del sistema con sus compañeros. Definidas ya las funcionalidades y los módulos del sistema, realizó pruebas junto a sus compañeros de las herramientas de modelado que Daniel iba proponiendo, pues era el que más familiarizado estaba con dichas herramientas. Entre todas las que existen, probó varias herramientas como Modelio [72], Enterprise Architect [42], Visual Paradigm [120], etc. Ante los continuos cambios en los diagramas, consensuó con su compañeros el uso de una herramienta más orientada al dibujo de diagramas que al desarrollo de los mismos. Esa herramienta fue draw.io [36], que se usó durante todo el resto del proyecto.

Por otro lado, participó en el estudio de los simuladores que se mencionan en los artículos [117, 98] y otros tales como SimGrid [99], PeerfactSim.KOM [85], etc. Al ver que la mayoría estaban obsoletos y no cumplían con las características que se necesitaban se optó por enfocar el proyecto hacia la

implementación. El lenguaje de programacion a usar sería Java [60].

Al dirigir el proyecto hacia la implementación se necesitaba elegir una red de sobrecapa con la que implementar el prototipo de la aplicación, por lo que participó junto a sus compañeros en la búsqueda de plataformas de tecnología P2P, como Chord [77] y FreePastry [44], y en las pruebas sobre una implementación muy básica de un chat. Estas pruebas consistieron, a grandes rasgos, en verificar la conexión al sistema por parte de los usuarios, envío de mensajes entre los mismos, etc.

En un primer acercamiento a lo que sería la implementación del prototipo de la aplicación, acordó junto a sus compañeros desarrollar una interfaz gráfica sencilla que permitiera llevar a cabo las acciones básicas posibles de un banco de tiempo. En un principio se determinó construir la vista del prototipo en Java Swing con WindowBuilder [143] y se decidió que sería el integrante del equipo que desarrollaría las vistas. Al poco tiempo de empezar con su desarrollo se decidió cambiar la herramienta a JavaFx [134], que permitía hacer unas vistas más profesionales y contemporáneas. Un tiempo después, una vez que hubo finalizado el desarrollo de las vistas, se acordó que éstas no iban a aparecer enganchadas al prototipo final por falta de tiempo, aunque sí se entregarían.

En el inicio de la redacción de la memoria, fue el encargado de investigar el sistema de composición de textos LaTeX [65], haciendo un pequeño esqueleto que permitiera iniciar la memoria con la portada, el índice, etc. Además asistió al curso *Apoyo a la preparación de memorias de SS. II., grado y máster: cómo documentar, citar y organizar mis referencias mediante las fuentes de información en Informática y los gestores bibliográficos*, el día 26 de noviembre de 2014 de 15:30 a 18:00 en el cual se explicó algunos aspectos de interés para redactar la memoria de una forma correcta, así como organizar las citas bibliográficas y cómo usarlas.

Durante la realización de los diagramas, además de ayudar al desarrollo de los mismos, fue el encargado de traducirlos a inglés y algunas partes de otros documentos, así como el nombre de las funciones.

Cabe destacar que también participó y aportó sus opiniones de manera activa en todo lo relacionado con el desarrollo y la implementación del pro-

totipo.

11.2. Daniel Alejandro Nowendsztern

En el inicio de este proyecto propuso una idea distinta de proyecto P2P de utilidad social. La idea era desarrollar una aplicación capaz de hacer *streaming* de videos en directo usando la cámara de los dispositivos móviles actuales. Para ese proyecto se usarían tecnologías punteras como Wi-Fi Direct [123] de Android [5]. Además cabría la posibilidad de añadirle geolocalización y otras mejoras en el futuro. Sin embargo la idea no se realizó. Esto fue debido a las dificultades para realizar un proyecto tan complejo en tan poco tiempo y al interés generado tras la propuesta de Antonio de hacer un banco de tiempo P2P.

Después de tomar la decisión de realizar la idea de un banco de tiempo P2P, la primera actividad que realizó fue la de proponer varios sistemas para alojar todo el material que se fuese encontrando en la red. Dos de los sistemas elegidos eran Google Drive [50] y Dropbox [37]. Finalmente se estableció como principal el primero.

Más adelante también propondría usar un repositorio para el control de versiones en Internet, dicho sistema era Google Code [49], aunque tras los cambios del mismo, decidiría optar por uno más estable, ese fue SourceForge [103].

Durante aquellas primeras semanas de proyecto también aportó algunos documentos importantes para especificar los requisitos y las funcionalidades del sistema. Algunos documentos eran los que había usado en proyectos de otras asignaturas de la carrera, por ejemplo el documento de especificación de requisitos software y el plan de proyecto.

Tras varias reuniones para decidir las funcionalidades del sistema, llegó la hora de empezar a definirlas. Fue entonces cuando, junto a sus compañeros, realizó la definición de los módulos y las funcionalidades del sistema, además de la especificación de los requisitos.

Con la definición de las funcionalidades definidas realizó una investigación sobre las posibles herramientas de modelado/dibujo útiles para realizar los diagramas. Entre todas las que existen, probó y propuso muchas herramientas como Modelio [72] (la conocía de años anteriores), Enterprise Architect [42], Visual Paradigm [120], etc.... Siempre en la búsqueda de mayor comodidad y eficiencia para realizar los diagramas que realizaría junto a sus compañeros. Sin embargo, ante los continuos cambios en los diagramas fue necesario que diese con una herramienta más gráfica que orientada al desarrollo, esa herramienta fue draw.io [36], que se usó durante todo el resto del proyecto.

En paralelo a la investigación de herramientas de modelado/dibujo, realizó la búsqueda de plataformas de tecnología P2P. Partiendo del desconocimiento total sobre posibles implementaciones de plataformas bajo la tecnología P2P, el primero que encontró fue Open Chord [77] que tras realizarle varias pruebas (tanto en red local como en Internet) sobre una implementación muy rudimentaria de un chat, comprobó los problemas que tenía tanto por la falta de documentación, como por la velocidad y el mantenimiento de la propia implementación.

Consideró usar una alternativa a esta plataforma, fue entonces cuando dió con FreePastry [44], que sin ser perfecto, ofrecía mayores ventajas que Open Chord (en documentación e implementación) y las posibilidades necesarias para implementar un prototipo de la aplicación. Además probó otras plataformas menos desarrolladas, no obtuvo mejores resultados.

A falta de unos meses para la finalización del proyecto, propusó una versión modificada del Eclipse IDE [40] con un plugin de LaTeX [65] para escribir junto a sus compañeros todo el contenido más relevante del proyecto en la memoria usando un repositorio con control de versiones.

Más adelante realizó el diseño de los logotipos del prototipo usando una versión de prueba de la herramienta de diseño gráfico Corel Draw X7 [25].

Finalmente realizó gran parte de la implementación del prototipo y de sus pruebas. Para realizar la implementación de una parte del prototipo del banco de tiempo, uso el lenguaje de programación Java [60]. La parte de la implementación en la que más profundizó fue la del *backend*, es decir, tanto

en la capa de los controladores como la capa del modelo de datos.

Cabe mencionar que también propusó algunas herramientas para facilitar la comunicación a distancia como Team Viewer [107] y que participó en otras actividades del proyecto, aunque en menor medida.

11.3. Marcos Pérez García

Fue el último integrante en incorporarse al grupo formado para abordar el proyecto. La propuesta de Antonio para realizar un banco de tiempo P2P ya se había acordado con el director, encontrandose el proyecto en el punto de partida.

Lo primero que hizo fue una investigación sobre el concepto de banco de tiempo, de qué se compone y qué ideas podían servir y ser interesantes para su desarrollo en P2P, proponiendo aplicar ideas tales como la oxidación del tiempo, la posibilidad de implantar un sistema de préstamos entre usuarios y de demandas de servicios.

En las primeras semanas de proyecto comenzó con la toma de requisitos del sistema y la especificación de las funcionalidades detectadas como claves junto al resto de sus compañeros.

Después de la definición del sistema se prosiguió con el modelado de la aplicación con multitud de herramientas que se fueron descartando, entre las que propuso un plugin de eclipse Papyrus [82], para al final elegir draw.io [36], una herramienta más gráfica que permitió abordar mejor los múltiples cambios que se dieron en los diagramas a lo largo del proyecto.

En un primero momento, el proyecto estaba dirigido a la creación de una simulación de un sistema P2P. Participó con el resto de sus compañeros en la búsqueda de los simularios P2P existentes, tales como PeerfactSim.KOM [85], PeerSim [86], etc. Pero la mayoría estaban obsoletos y fue necesario enfocar el proyecto hacia un desarrollo del sistema.

Después de consensuar con el grupo que la tecnología que se usaría para

la implementación sería Java [60], realizó una búsqueda de las primeras plataformas Open Chord [77] y FreePasty [44] para tratar de asimilar su funcionamiento y poder utilizarlos en la implementación del prototipo. Open Chord fue el primero con el que se experimiento, pero después de crear un ejemplo de chat muy básico se decidió descartar principalmente por problemas en su funcionamiento, la falta de documentación y la existencia de ejemplos que pudieran servir como punto de partida. Con FreePasty las sensaciones fueron mejores durante las pruebas de algunos ejemplos, aunque también se extrañaba una mejor documentación. Finalmente el elegido fue este último.

Para la creación del diseño de la interfaz del prototipo del banco de tiempo, propuso una base tomada de un software para bancos de tiempo en España, TimeOverflow [112]. Este software contenía unos *wireframes* muy interesantes de los que se podían sacar ideas que encajaban con la especificación que se había hecho previamente.

Durante los últimos meses del proyecto, la planificación y el control se hicieron muy complejos de llevar simplemente con Google Drive [50], Dropbox [37] y las reunniones que se hacían periódicamente. En ese punto, existía mucho material y muchas tareas abiertas que finalizar. Fue entonces cuando el director del proyecto comento la posibilidad de investigar y utilizar una herramienta más concreta para la planificación y el seguimiento de proyectos como Redmine [95]. Se encargó de realizar una búsqueda de esta herramienta en una versión *online* y gratuita que pudiera ser accesible para todo el grupo [54]. De esta forma pudo dar de alta las tareas pendientes y plantear un calendario (diagrama de Gantt [128]) con las tareas previstas, distribuidas en el tiempo que quedaba disponible antes de la finalización del proyecto, así como de mantener esta planificación.

Además, al mismo tiempo que se introdujo Redmine, la frecuencia de las reuniones se incremento por lo que propuso la creación de actas de reunión para una mejor organización y completar así un buen seguimiento y control de los hitos intermedios que se acordaban.

También realizó una investigación de los distintos tipos de criptografía, el funcionamiento de una firma digital y la generación de un par de claves pública y privada para poder aplicarlo en la implementación del prototipo. Cabe destacar que, desarrolló una primera versión de la parte del modelo del

prototipo e intervino en otras múltiples partes de la implementación que se abordaron en grupo. También es de especial interés mencionar que insistió en la importancia de generar un Javadoc [61], para crear así un software más completo sin los problemas de documentación detectados en la mayoría de las plataformas y simuladores P2P encontrados.

Por último, utilizó un plugin de LaTeX [65] para Eclipse IDE [40], propuesto por Daniel, para escribir el contenido de la memoria junto al resto de sus compañeros.

Apéndice A

Especificación de los requisitos

A.0.1. Subsistema básico o de gestión de usuarios

Gestión de usuarios

- **Función:** Alta de usuario.
  ID: Usu01.
  Prioridad: Alta.
  Estabilidad: Alta.
  Descripción: Dar de alta un nuevo usuario en el sistema.
  Entrada: Datos del nuevo usuario.
  Salida: Mensaje de confirmación de alta realizada.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red.
  Acción: Se comprueba el UUID del usuario. En caso de que exista un UUID con ese nombre de usuario en local devolvería error. En caso contrario se genera el UUID, las claves y el perfil y se almacenan en la red P2P.
  Precondición: Datos de nuevo usuario válidos.
  Postcondición: Nuevo usuario introducido en el sistema.

- **Función:** Baja de usuario.
  ID: Usu02.
  Prioridad: Alta.
  Estabilidad: Alta.

Descripción: Dar de baja un nuevo usuario en el sistema.
Entrada: Datos del usuario.
Salida: Mensaje de confirmación de baja realizada.
Origen: Interfaz gráfica.
Destino: Sistema.
Necesita: Estar conectado a la red y con sesión iniciada.
Acción: Busca los perfiles público y privado en la red, los deshabilita y los almacena de nuevo en la red. Finalmente, se desconecta de la red.
Precondición: Usuario existente en el sistema.
Postcondición: Usuario eliminado del sistema.

- **Función:** Editar datos de usuario.
  ID: Usu03.
  Prioridad: Baja.
  Estabilidad: Alta.
  Descripción: Editar datos de un usuario en el sistema.
  Entrada: Datos del usuario.
  Salida: Mensaje de confirmación de modificación realizada.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: Busca los perfiles público y privado en la red, los modifica y los almacena de nuevo en la red.
  Precondición: Datos de usuario válidos.
  Postcondición: Usuario modificado en el sistema.

- **Función:** Iniciar sesión.
  ID: Usu04.
  Prioridad: Alta.
  Estabilidad: Alta.
  Descripción: Iniciar sesión en el sistema.
  Entrada: Datos del usuario.
  Salida: -.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y registrado en el sistema.

Acción: Busca el UUID del usuario correspondiente en local. En caso de que no exista devuelve error. Si existe busca el perfil privado para comprobar si la contraseña es correcta y si la cuenta está activa. Si la contraseña no es correcta o la cuenta no está activa entonces devuelve error. En caso contrario se inicia sesión.
Precondición: Datos de nuevo usuario válidos y usuario existente en el sistema.
Postcondición: Sesión iniciada en el sistema.

- **Función:** Cerrar sesión.
  ID: Usu05.
  Prioridad: Alta.
  Estabilidad: Alta.
  Descripción: Cerrar sesión en el sistema.
  Entrada: Datos del usuario.
  Salida: -.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: Se avisa a las capas inferiores de que el nodo se va a desconectar para hacer una salida ordenada. Se muestra la pantalla de login.
  Precondición: Usuario con sesión iniciada en el sistema.
  Postcondición: Sesión cerrada en el sistema.

- **Función:** Ver perfil de usuario.
  ID: Usu06.
  Prioridad: Media.
  Estabilidad: Media.
  Descripción: Ver perfil del usuario en el sistema.
  Entrada: -.
  Salida: Los datos del usuario.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: Se descarga el perfil de usuario y se muestran sus datos.
  Precondición: Usuario con sesión iniciada en el sistema.

Postcondición: -.

A.0.2. Subsistema de transacciones

Gestión de facturas

- **Función:** Enviar presupuesto.
  ID: Fac01.
  Prioridad: Alta.
  Estabilidad: Alta.
  Descripción: Envía un presupuesto a otro usuario del sistema.
  Entrada: Presupuesto y destinatario.
  Salida: Mensaje de confirmación de presupuesto enviado.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: El acreedor inicia la negociación creando un presupuesto, lo almacena en la red sin firmar y notifica al deudor.
  Precondición: Presupuesto firmado y destinatario válido.
  Postcondición: Presupuesto enviado.

- **Función:** Enviar factura proforma.
  ID: Fac02.
  Prioridad: Alta.
  Estabilidad: Alta.
  Descripción: Envía una factura proforma a otro usuario del sistema.
  Entrada: Factura proforma y destinatario.
  Salida: Mensaje de confirmación de factura proforma enviada.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: El acreedor continúa la negociación creando una factura proforma, la almacena en la red sin firmar y notifica al deudor.
  Precondición: Factura proforma firmada y destinatario válido.
  Postcondición: Factura proforma enviada.

- **Función:** Enviar factura.
  ID: Fac03.
  Prioridad: Alta.
  Estabilidad: Alta.
  Descripción: Envía una factura a otro usuario del sistema.
  Entrada: Factura y destinatario.
  Salida: Mensaje de confirmación de factura enviada.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: El acreedor finaliza la negociación creando una factura, la almacena en la red sin firmar y notifica al deudor.
  Precondición: Factura firmada y destinatario válido.
  Postcondición: Factura enviada.

- **Función:** Enviar factura.
  ID: Fac04.
  Prioridad: Alta.
  Estabilidad: Alta.
  Descripción: Envía una factura a otro usuario del sistema.
  Entrada: Factura y destinatario.
  Salida: Mensaje de confirmación de factura enviada.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: El acreedor finaliza la negociación creando una factura, la almacena en la red sin firmar y notifica al deudor.
  Precondición: Factura firmada y destinatario válido.
  Postcondición: Factura enviada.

- **Función:** Ver presupuesto.
  ID: Fac05.
  Prioridad: Alta.
  Estabilidad: Media.
  Descripción: Se muestra el presupuesto.
  Entrada: Notificación del acreedor.

Salida: Los datos del presupuesto.
Origen: Interfaz gráfica.
Destino: Sistema.
Necesita: Estar conectado a la red y con sesión iniciada.
Acción: El deudor recibe la notificación del acreedor, descarga el presupuesto de la red, comprueba su validez y lo muestra.
Precondición: Presupuesto válido (firmado digitalmente).
Postcondición: Presupuesto mostrado.

- **Función:** Ver factura proforma.
  ID: Fac06.
  Prioridad: Alta.
  Estabilidad: Media.
  Descripción: Se muestra la factura proforma.
  Entrada: Notificación del acreedor.
  Salida: Los datos de la factura proforma.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: El deudor recibe la notificación del acreedor, descarga la factura proforma de la red, comprueba su validez y lo muestra.
  Precondición: Factura proforma válida (firmada digitalmente).
  Postcondición: Factura proforma mostrada.

- **Función:** Ver factura.
  ID: Fac07.
  Prioridad: Alta.
  Estabilidad: Media.
  Descripción: Se muestra la factura.
  Entrada: Notificación del acreedor.
  Salida: Los datos de la factura.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: El deudor recibe la notificación del acreedor, descarga la factura de la red, comprueba su validez y lo muestra.

Precondición: Factura válida (firmada digitalmente).
Postcondición: Factura mostrada.

- **Función:** Ver transacciones en curso.
  ID: Fac08.
  Prioridad: Alta.
  Estabilidad: Media.
  Descripción: Se muestran las transacciones en curso.
  Entrada: Perfil privado.
  Salida: Los datos de las transacciones en curso.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: El usuario descarga el perfil privado de la red y obtiene la lista de transacciones en curso.
  Precondición: Perfil privado válido (firmado digitalmente).
  Postcondición: Transacciones en curso mostradas.

- **Función:** Ver transacciones en finalizadas.
  ID: Fac09.
  Prioridad: Alta.
  Estabilidad: Media.
  Descripción: Se muestran las transacciones finalizadas.
  Entrada: Perfil privado.
  Salida: Los datos de las transacciones finalizadas.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: El usuario descarga el perfil privado de la red y obtiene la lista de transacciones finalizadas.
  Precondición: Perfil privado válido (firmado digitalmente).
  Postcondición: Transacciones finalizadas mostradas.

- **Función:** Efectuar pago.
  ID: Fac10.
  Prioridad: Alta.

Estabilidad: Media.
Descripción: Se efectúa el pago tras la realización de un servicio ofertado.
Entrada: Datos de la factura.
Salida: Notificación de éxito/fallo.
Origen: Interfaz gráfica.
Destino: Sistema.
Necesita: Estar conectado a la red y con sesión iniciada.
Acción: El deudor inicia el pago descargando la factura. Genera los ficheros necesarios para efectuar el pago y notifica al acreedor. El acreedor recibe la notificación y descarga los ficheros creados por el deudor y comprueba su validez. En caso de no ser válidos devuelve error.
El acreedor modifica los ficheros descargados (puede agregar un comentario), los firma, los almacena en la red y notifica al deudor. El deudor recibe la notificación, descarga los ficheros modificados y los comprueba. En caso de no ser válidos devuelve error.
A continuación, el deudor modifica los ficheros descargados y añade sus entradas finales (puede agregar un comentario), los firma, los almacena en la red y notifica al acreedor. El acreedor recibe la notificación, descarga los ficheros modificados y los comprueba. En caso de no ser válidos devuelve error.
Por último, el acreedor modifica los ficheros descargados y añade sus entradas finales, los firma, los almacena en la red y notifica al deudor. El deudor recibe la notificación, descarga los ficheros modificados y los comprueba. En caso de no ser válidos devuelve error.
Precondición: Factura existente.
Postcondición: Pago realizado.

Gestión de reputación

- **Función:** Escribir comentario y valorar servicio.
  ID: Rep01.
  Prioridad: Media.
  Estabilidad: Media.
  Descripción: Permitir escribir comentarios a los usuarios sobre los servicios prestados/recibidos.
  Entrada: Comentario y valoración.

Salida: Mensaje de confirmación.
Origen: Interfaz gráfica.
Destino: Sistema.
Necesita: Estar conectado a la red y con sesión iniciada.
Acción: Escribir un comentario sobre un servicio y valorarlo.
Precondición: Que el servicio se haya realizado.
Postcondición: Comentario del usuario agregado.

- **Función:** Mostrar comentario y/o valoración.
  ID: Rep02.
  Prioridad: Baja.
  Estabilidad: Media.
  Descripción: Permite a un usuario ver un comentario.
  Entrada: ID del comentario.
  Salida: Mensaje de confirmación.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: Editar un comentario y/o valoración de un servicio.
  Precondición: Comentario y/o valoración existentes.
  Postcondición: Comentario y/o valoración mostrados.

- **Función:** Editar comentario y/o valoración.
  ID: Rep03.
  Prioridad: Baja.
  Estabilidad: Media.
  Descripción: Permite a un usuario editar su propio comentario almacenado en local dentro de un plazo de 30 minutos antes de ser enviado a la red.
  Entrada: ID del comentario.
  Salida: Mensaje de confirmación.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: Editar un comentario y/o valoración de un servicio.
  Precondición: Comentario y valoración existentes.

Postcondición: Comentario y valoración actualizados.

Gestión de balance/histórico

- **Función:** Ver actividades realizadas.
  ID: Bal01.
  Prioridad: Media.
  Estabilidad: Media.
  Descripción: Se muestran los registros de todas las transacciones realizadas.
  Entrada: Datos de usuario.
  Salida: Lista detallada con todas las transacciones realizadas.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y sesión iniciada.
  Acción: El sistema muestra todas las transacciones realizadas.
  Precondición: Usuario existente.
  Postcondición: Actividades realizadas mostradas.

Gestión de préstamos

- **Función:** Hacer préstamo.
  ID: Pre01.
  Prioridad: Baja.
  Estabilidad: Media.
  Descripción: Permitir realizar un préstamo de horas entre usuarios del sistema.
  Entrada: Información del usuario que realiza el préstamo, información del usuario que recibe el préstamo y datos del préstamo.
  Salida: Mensaje de confirmación de préstamo realizado.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: Disminuye el crédito del usuario que realiza el préstamo mientras que aumenta el crédito del usuario que recibe el préstamo en el perfil del usuario. Se genera una deuda del usuario receptor hacia el

usuario emisor.
Precondición: Los dos usuarios existen en el sistema. Número de horas a prestar no sea superior al crédito del usuario que presta.
Postcondición: Préstamo creado.

- **Función:** Solicitar préstamo.
  ID: Pre02.
  Prioridad: Baja.
  Estabilidad: Media.
  Descripción: Permitir solicitar un préstamo de horas entre usuarios del sistema.
  Entrada: Información del usuario que realiza el préstamo, información del usuario que recibe el préstamo y datos del préstamo.
  Salida: Mensaje de confirmación de préstamo solicitado.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: Solicita un préstamo a un usuario.
  Precondición: Usuario prestamista existente.
  Postcondición: Solicitud de préstamo enviada.

- **Función:** Mostrar préstamo.
  ID: Pre03.
  Prioridad: Baja.
  Estabilidad: Media.
  Descripción: Muestra un préstamo de horas entre usuarios del sistema.
  Entrada: Datos del préstamo.
  Salida: Datos del préstamo.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: Muestra un préstamo.
  Precondición: Préstamo existente. Número de horas a prestar no sea superior al crédito del usuario que presta.
  Postcondición: Datos del préstamo mostrados.

A.0.3. Subsistema de servicios

Gestión de servicios

- **Función:** Ofertar servicio.
  ID: Ser01.
  Prioridad: Alta.
  Estabilidad: Alta.
  Descripción: Oferta un servicio para el banco de tiempo.
  Entrada: Datos del servicio.
  Salida: Mensaje de confirmación de servicio dado de alta.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: Dar de alta un servicio nuevo.
  Precondición: Usuario que quiere ofrecer el servicio existe en el sistema.
  Postcondición: Nuevo servicio introducido en el sistema.

- **Función:** Mostrar servicio.
  ID: Ser02.
  Prioridad: Alta.
  Estabilidad: Alta.
  Descripción: Muestra un servicio para el banco de tiempo.
  Entrada: Datos de servicio.
  Salida: Datos del servicio.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: Mostrar un servicio.
  Precondición: Servicio existente en el sistema.
  Postcondición: Datos del servicio mostrado.

- **Función:** Demandar servicio.
  ID: Ser03.

Prioridad: Alta.
Estabilidad: Media.
Descripción: Demanda un servicio.
Entrada: Datos del servicio.
Salida: Mensaje de confirmación.
Origen: Interfaz gráfica.
Destino: Sistema.
Necesita: Estar conectado a la red y con sesión iniciada.
Acción: Se almacena una nueva demanda de servicio.
Precondición: Datos del servicio.
Postcondición: Demanda de servicio creada.

- **Función:** Buscar servicio.
  ID: Ser04.
  Prioridad: Alta.
  Estabilidad: Media.
  Descripción: Realizar una consulta sobre un tipo de oferta en la red.
  Entrada: Datos de la oferta.
  Salida: Lista de ofertas que encajan con la consulta.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: El sistema devuelve una lista de ofertas.
  Precondición: Datos de la oferta.
  Postcondición: Listado de ofertas.

- **Función:** Cancelar servicio.
  ID: Ser05.
  Prioridad: Alta.
  Estabilidad: Media.
  Descripción: Se cancela el servicio del sistema.
  Entrada: Datos del servicio.
  Salida: Notificación de éxito/fallo.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.

Acción: El sistema inhabilita el servicio.
Precondición: Servicio existente.
Postcondición: Servicio inhabilitado.

- **Función:** Enviar solicitud de servicio.
  ID: Ser06.
  Prioridad: Alta.
  Estabilidad: Alta.
  Descripción: Enviar una solicitud para recibir un servicio.
  Entrada: El servicio y datos de los usuarios implicados.
  Salida: Mensaje de confirmación.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: El usuario deudor envía una notificación al acreedor con los datos de entrada correspondientes.
  Precondición: Que los dos usuarios existan en el sistema, que el servicio exista en el sistema.
  Postcondición: Solicitud de servicio enviada.

- **Función:** Aceptar solicitud de servicio.
  ID: Ser07.
  Prioridad: Alta.
  Estabilidad: Media.
  Descripción: El acreedor acepta una solicitud de servicio.
  Entrada: Solicitud de servicio.
  Salida: Mensaje de confirmación.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: El usuario acreedor envía una notificación al deudor con los datos de entrada correspondientes.
  Precondición: Servicio existente.
  Postcondición: Solicitud de servicio aceptada.

- **Función:** Mostrar solicitud de servicio.

ID: Ser08.
Prioridad: Alta.
Estabilidad: Media.
Descripción: Se muestra una solicitud de servicio por parte del deudor.
Entrada: Datos de la solicitud del servicio.
Salida: Datos de solicitud del servicio.
Origen: Interfaz gráfica.
Destino: Sistema.
Necesita: Estar conectado a la red y con sesión iniciada.
Acción: Se muestran los datos de la solicitud del servicio.
Precondición: Servicio existente.
Postcondición: Datos de solicitud del servicio presentados.

- **Función:** Cancelar solicitud de servicio.
  ID: Ser09.
  Prioridad: Alta.
  Estabilidad: Media.
  Descripción: Se rechaza una solicitud de servicio por parte del deudor.
  Entrada: Datos de la solicitud del servicio.
  Salida: Mensaje de confirmación de servicio cancelado.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: Se cancela la solicitud de servicio.
  Precondición: Servicio existente.
  Postcondición: Solicitud de servicio cancelada.

Gestión de ontología

- **Función:** Crear una categoría / clasificación.
  ID: Ont01.
  Prioridad: Alta.
  Estabilidad: Media.
  Descripción: Creación de una categoría / clasificación para la agru-

pación de servicios.
Entrada: Datos de la categoría: Nombre de la categoría.
Salida: Mensaje de confirmación de creación de categoría realizada.
Origen: Interfaz gráfica.
Destino: Sistema.
Necesita: Estar conectado a la red y con sesión iniciada.
Acción: Dar de alta una categoría / clasificación nueva.
Precondición: Usuario existente y categoría no existente.
Postcondición: Nueva categoría / clasificación introducida en el sistema.

- **Función:** Mostrar una categoría / clasificación.
  ID: Ont02.
  Prioridad: Alta.
  Estabilidad: Media.
  Descripción: Muestra una categoría / clasificación de servicios.
  Entrada: Datos de la categoría / clasificación.
  Salida: Categoría / clasificación.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: Mostrar una categoría / clasificación.
  Precondición: Categoría / clasificación existente.
  Postcondición: Categoría / clasificación mostrada.

A.0.4. Subsistema de notificaciones

Gestión de notificaciones

- **Función:** Enviar notificación.
  ID: Not01.
  Prioridad: Alta.
  Estabilidad: Media.
  Descripción: Envía una notificación a otro usuario.
  Entrada: Datos de la notificación.
  Salida: Mensaje de confirmación de notificación enviada.
  Origen: Interfaz gráfica.

Destino: Sistema.
Necesita: Estar conectado a la red y con sesión iniciada.
Acción: Enviar un mapeo del ID del fichero con la dirección en la DHT.
Precondición: Usuario destinatario existe en el sistema.
Postcondición: Notificación enviada en el sistema.

- **Función:** Leer notificación.
  ID: Not02.
  Prioridad: Alta.
  Estabilidad: Media.
  Descripción: Lee la notificación de otro usuario.
  Entrada: Datos de la notificación.
  Salida: Mensaje de confirmación de notificación recibida.
  Origen: Interfaz gráfica.
  Destino: Sistema.
  Necesita: Estar conectado a la red y con sesión iniciada.
  Acción: Leer un mapeo del ID del fichero con la dirección en la DHT.
  Precondición: Datos de la notificación verificados.
  Postcondición: Datos de la notificación mostrados.

Apéndice B

Documentos de análisis

B.1. Diagramas de casos de uso

Cada uno de los diagramas de casos de uso que se muestran a continuación hace referencia a un módulo concreto del sistema.

B.1.1. Subsistema básico o de gestión de usuarios

Gestión de usuarios

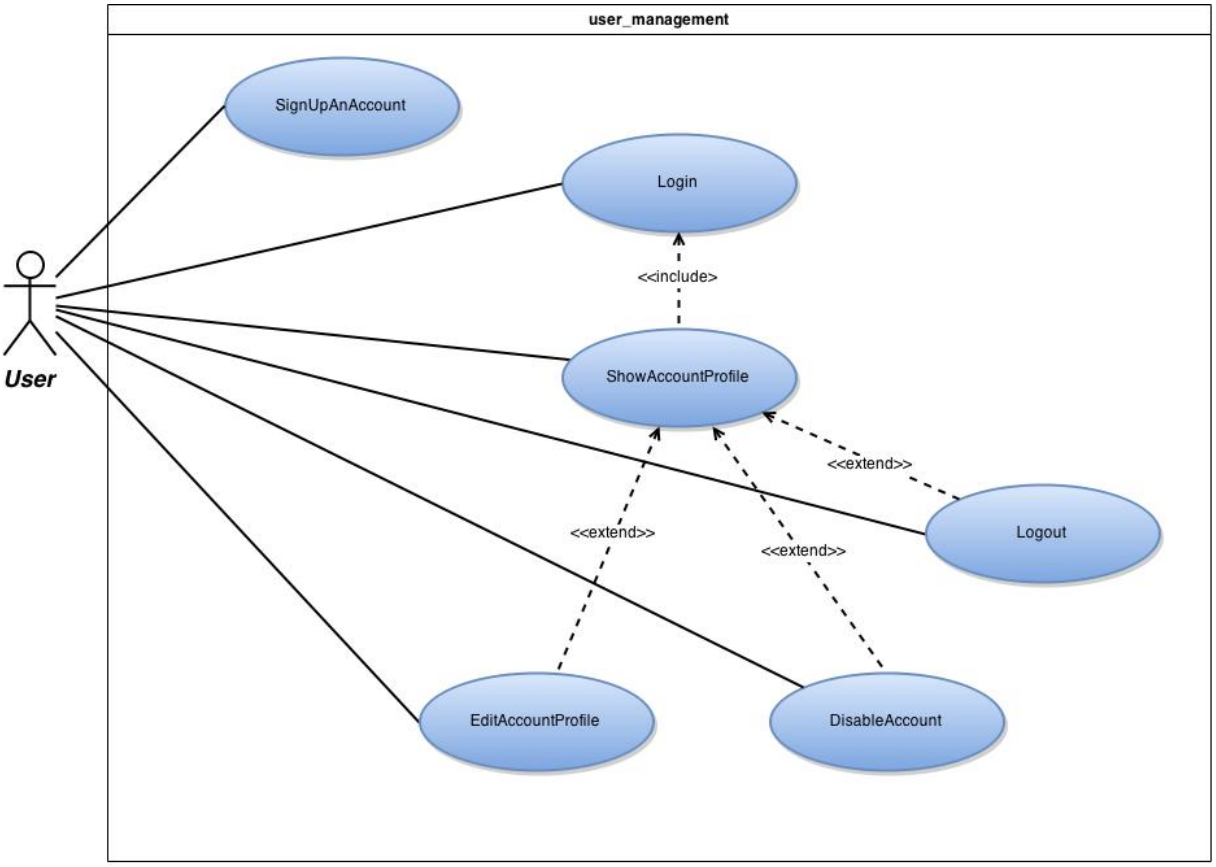

Figura B.1: Diagrama de casos de uso de gestión de usuarios.

B.1.2. Subsistema de transacciones

Gestión de facturas

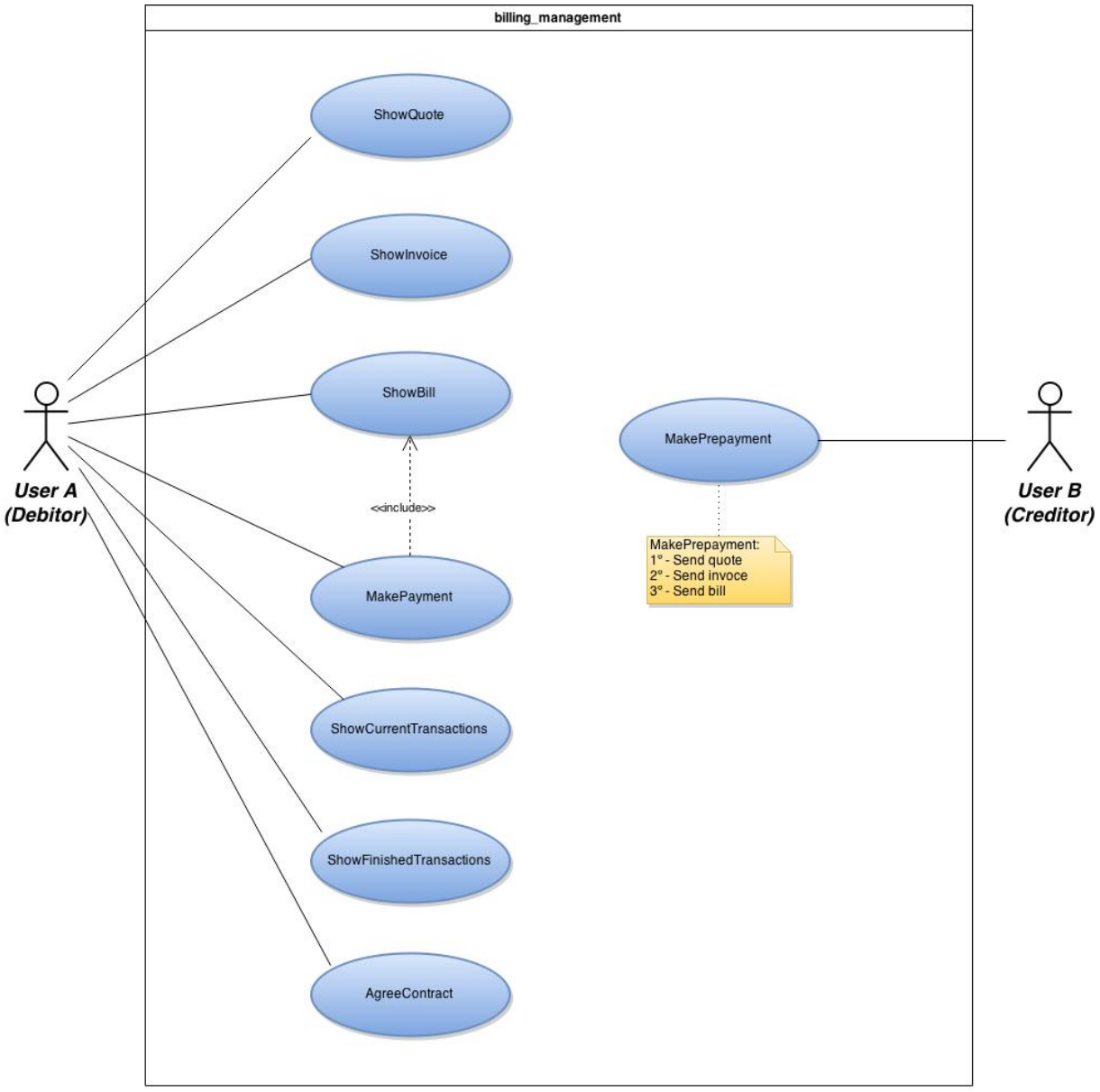

Figura B.2: Diagrama de casos de uso de gestión de facturas.

Gestión de reputación

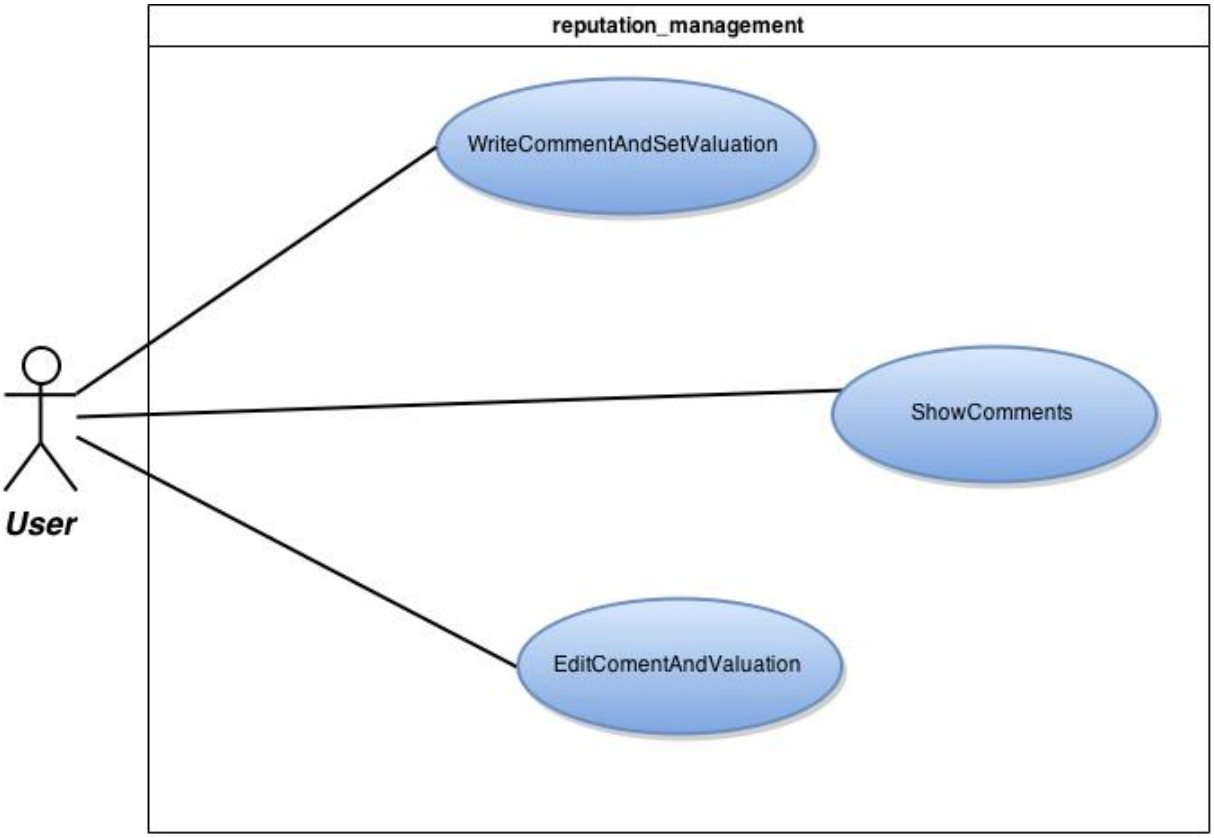

Figura B.3: Diagrama de casos de uso de gestión de reputación.

Gestión de balance/histórico

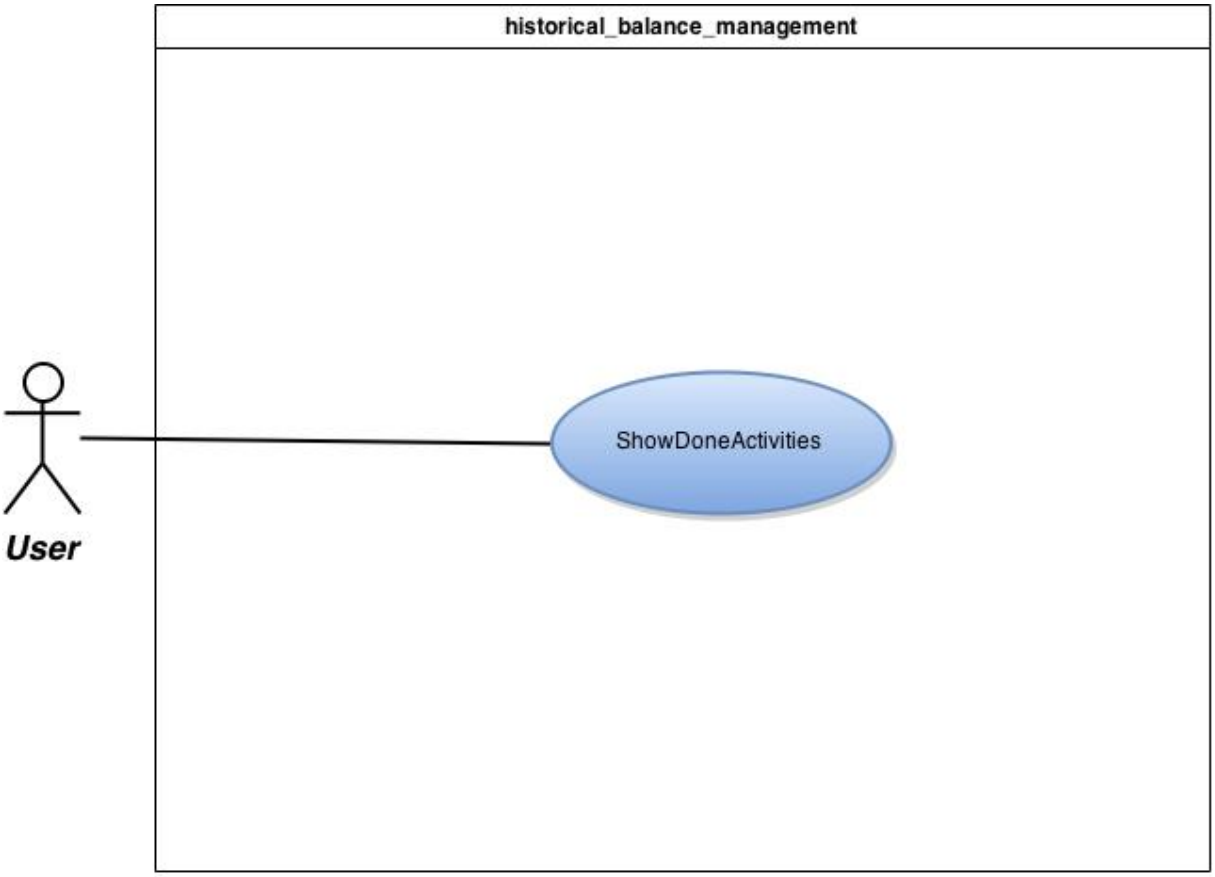

Figura B.4: Diagrama de casos de gestión de balance/histórico.

Gestión de préstamos

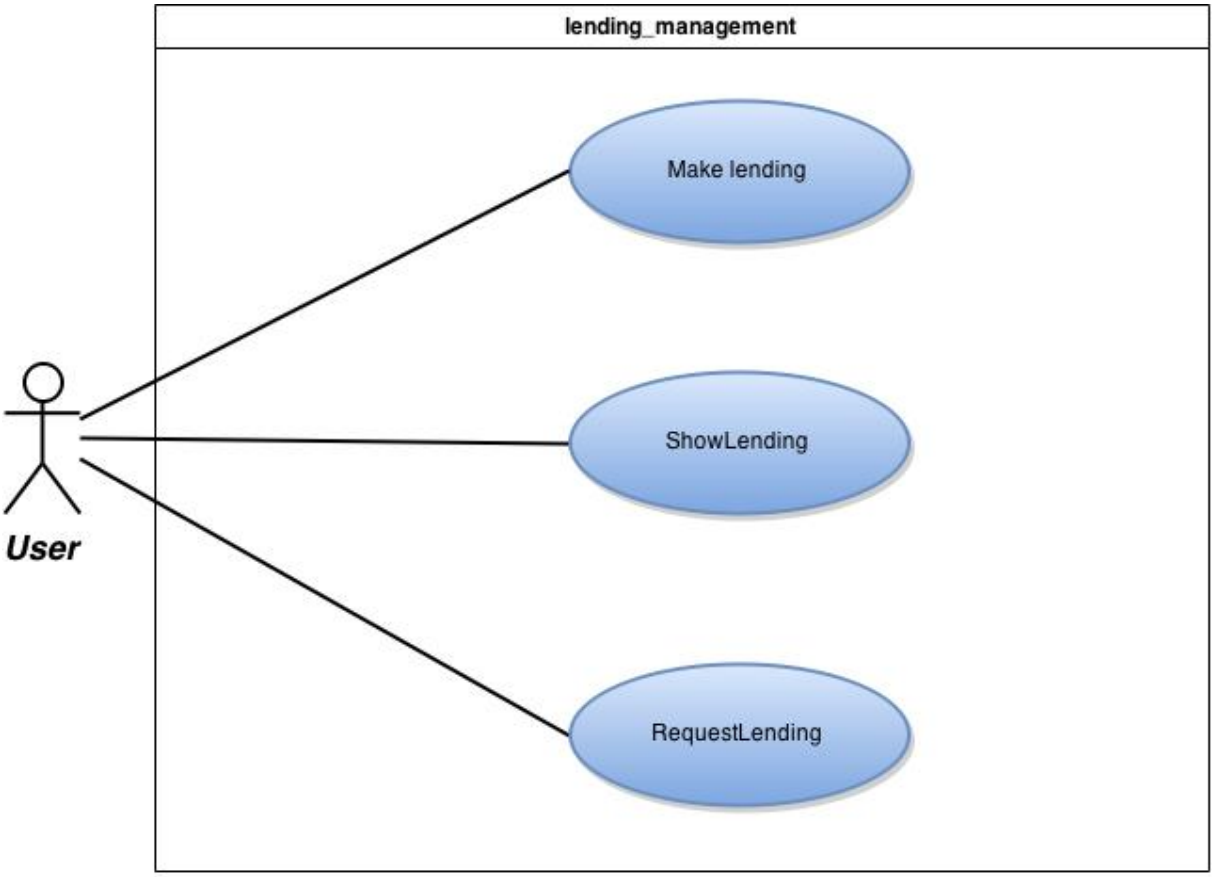

Figura B.5: Diagrama de casos de uso de gestión de préstamos.

B.1.3. Subsistema de servicios

Gestión de servicios

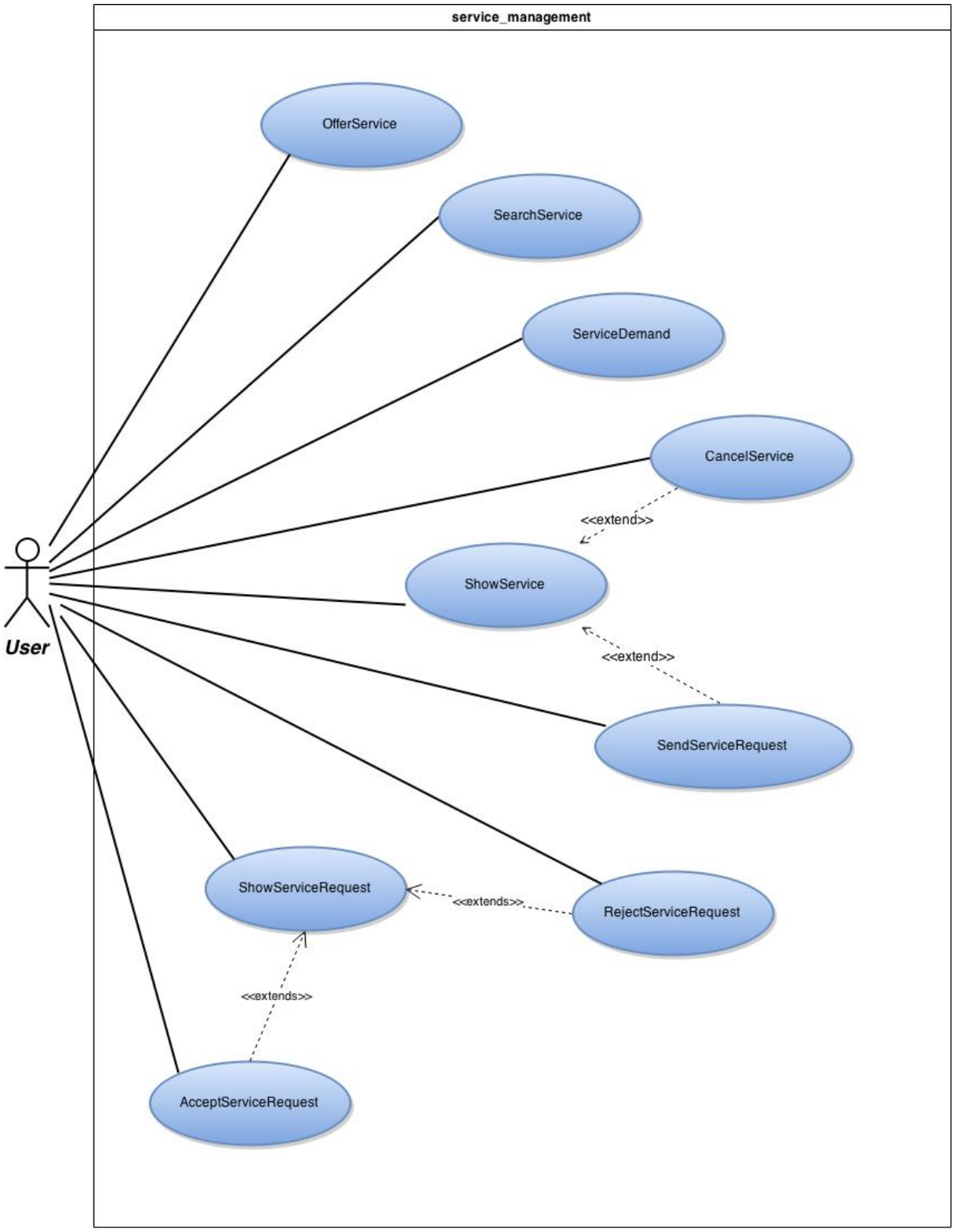

Figura B.6: Diagrama de casos de uso de gestión de servicios.

Gestión de ontología

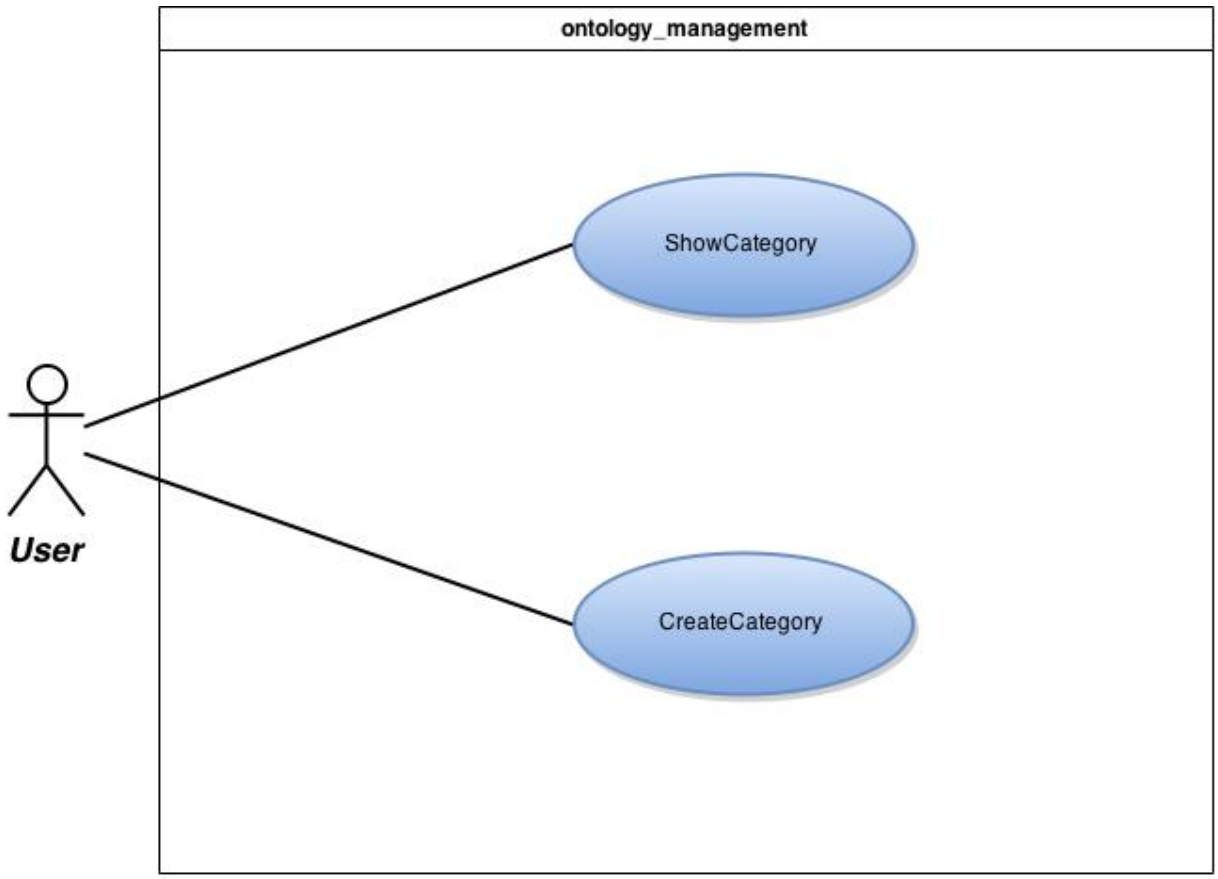

Figura B.7: Diagrama de casos de uso de gestión de ontología.

B.1.4. Subsistema de notificaciones

Gestión de notificaciones

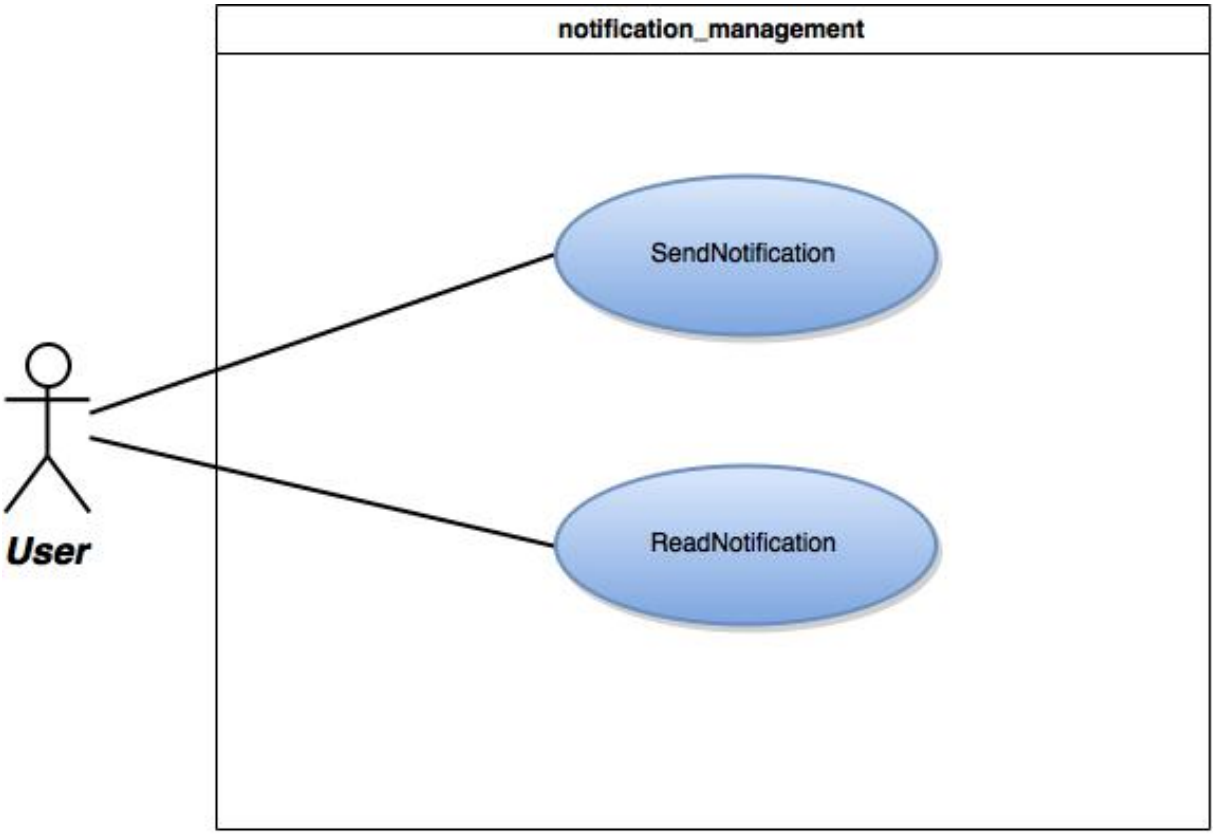

Figura B.8: Diagrama de casos de uso de gestión de notificaciones.

B.2. Diagramas de clase

Como una idea general para representar la estructura de objetos que componen el sistema se definió el modelo de dominio que aparece en la figura B.9.

A partir de ese modelo de dominio se realizaron los siguientes diagramas de clases que entran ya en lo que es el diseño del prototipo del sistema.

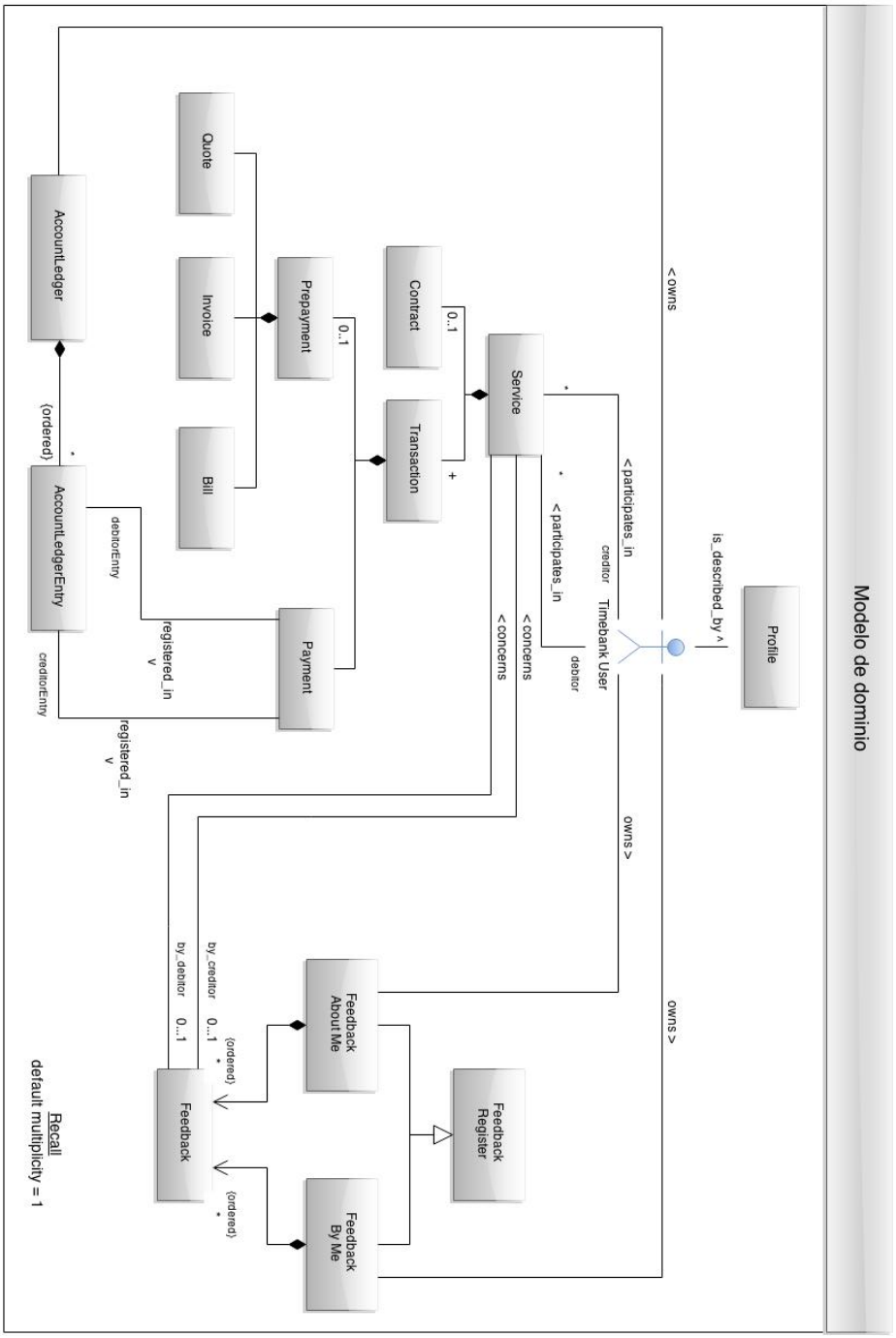

Figura B.9: Modelo de dominio.

B.3. Diagramas de secuencia

Los diagramas de secuencia que se muestran a continuación describen los procedimientos más complejos e interesantes que tiene el sistema.

B.3.1. Subsistema básico o de gestión de usuarios

Gestión de usuarios

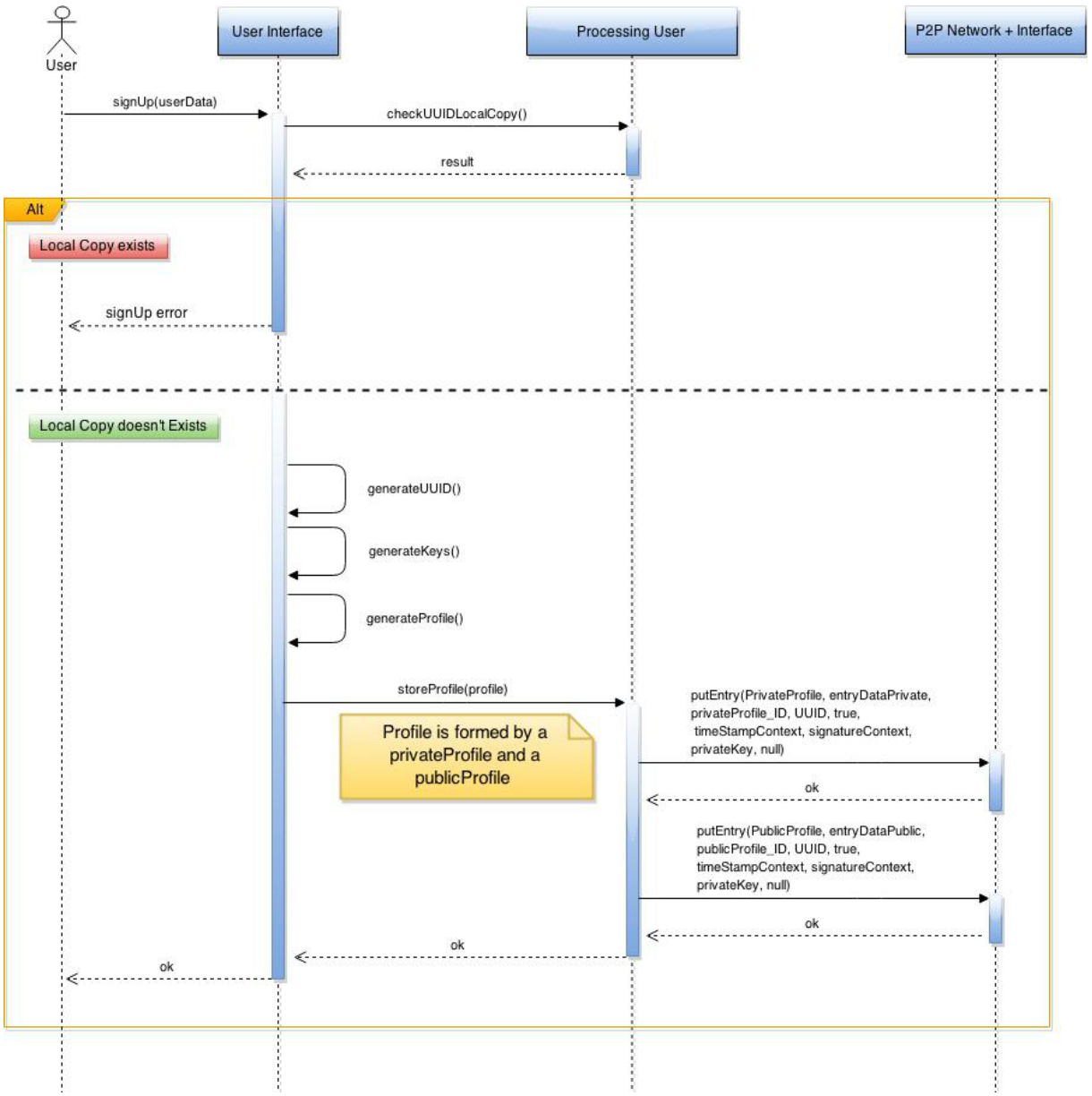

Figura B.10: Diagrama de secuencia del alta de usuario.

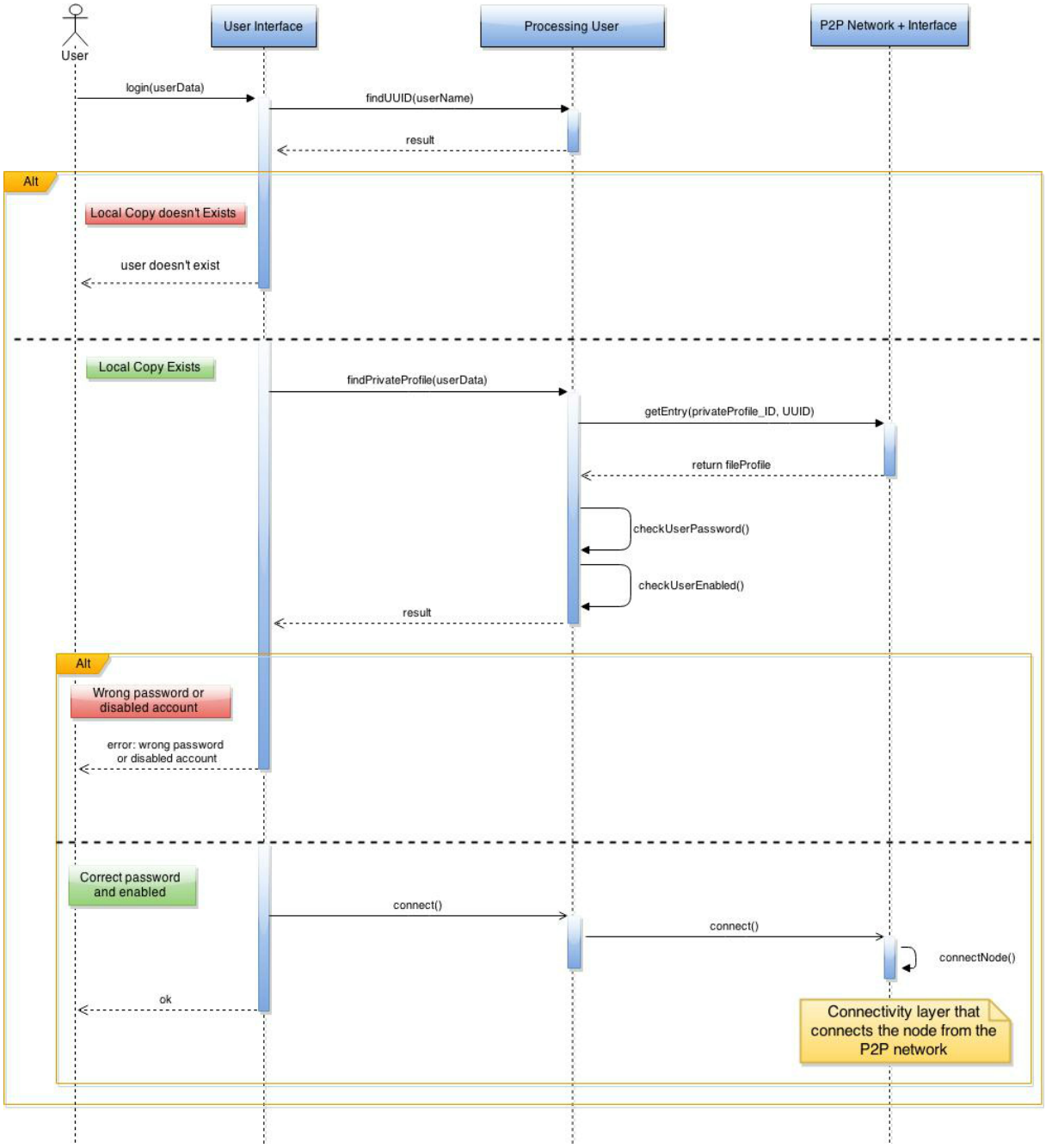

Figura B.11: Diagrama de secuencia del login de usuario.

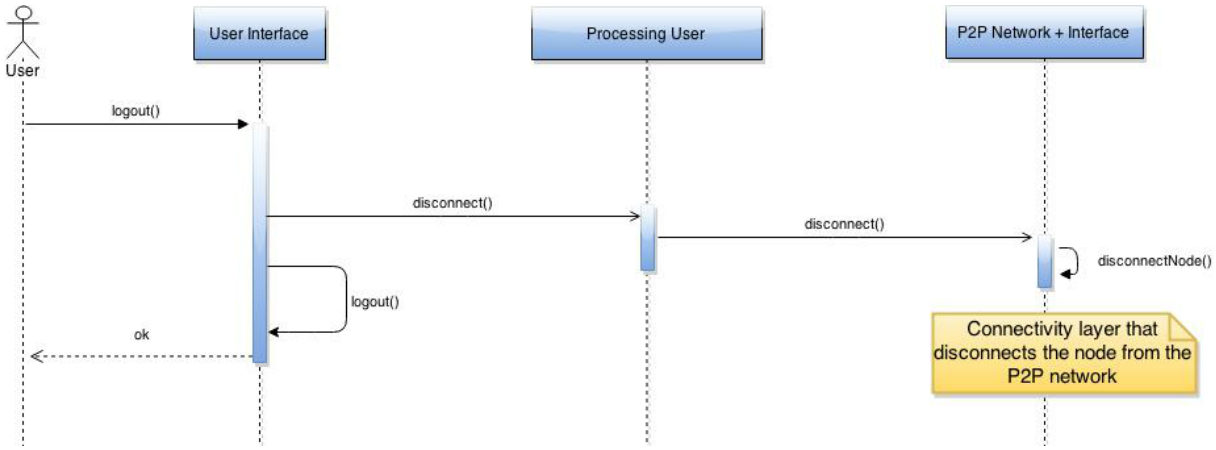

Figura B.12: Diagrama de secuencia del logout de usuario.

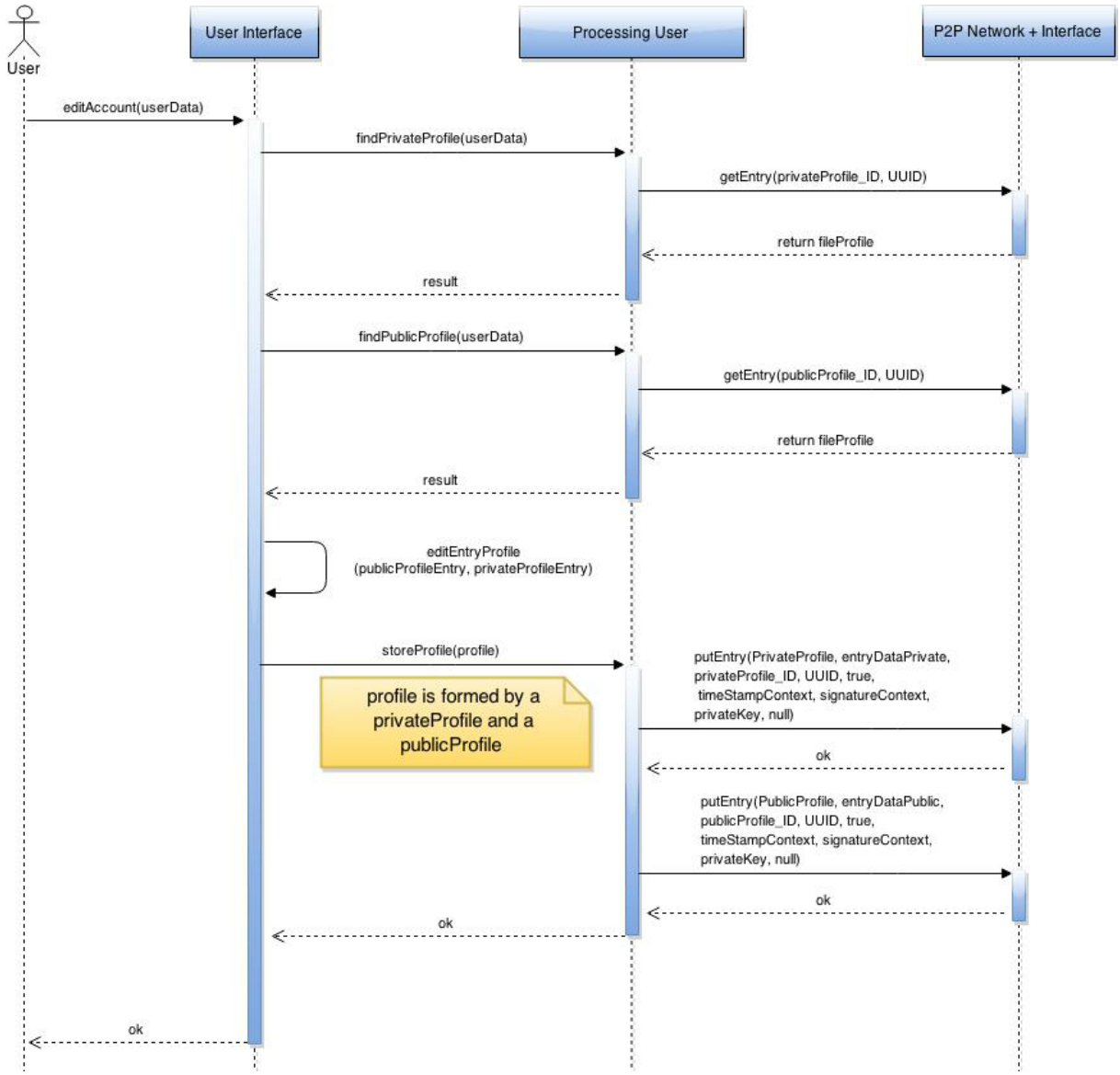

Figura B.13: Diagrama de secuencia de edición de usuario.

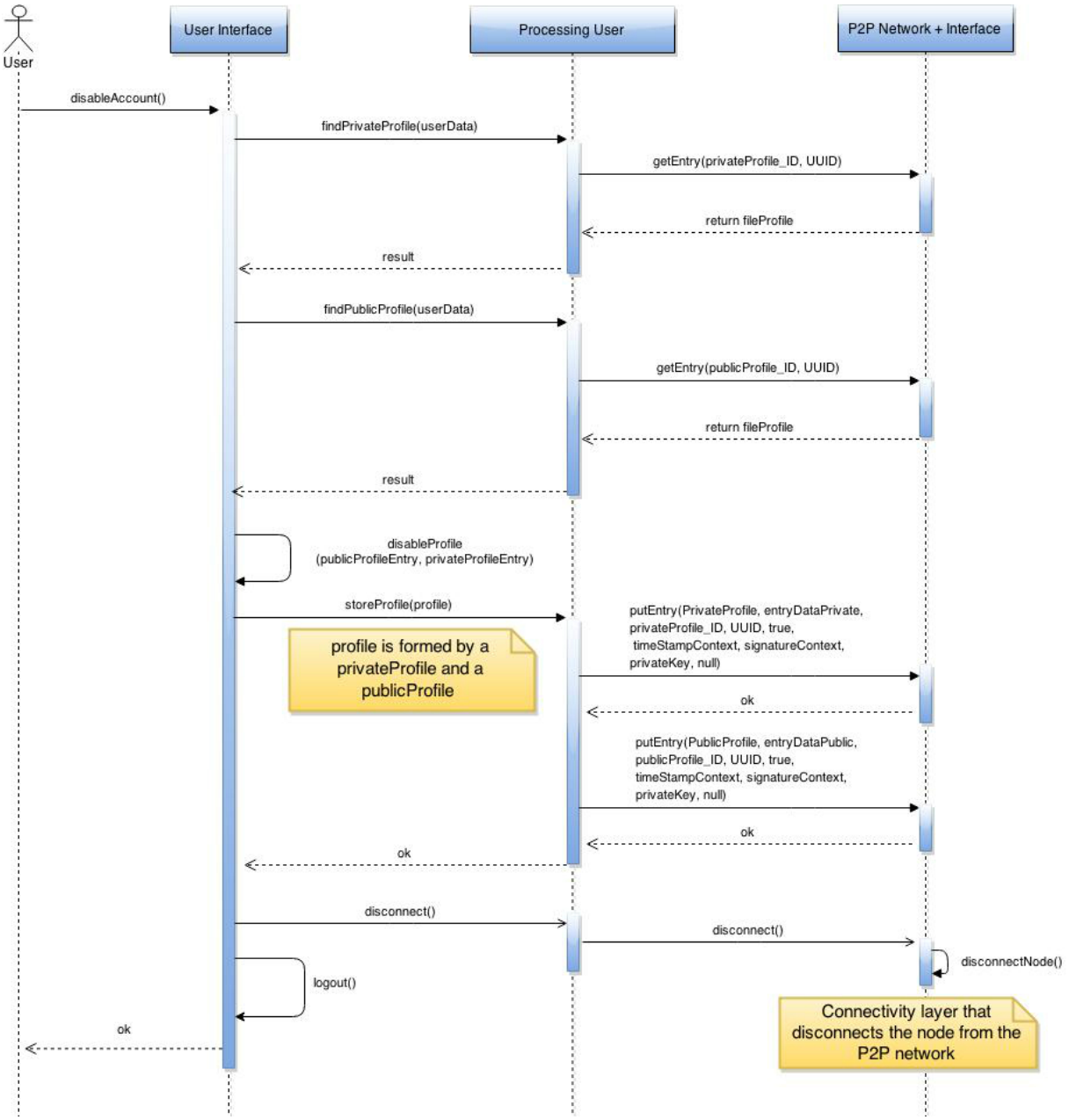

Figura B.14: Diagrama de secuencia de dar de baja a un usuario.

B.3.2. Subsistema de transacciones

Gestión de facturas

Efectuar pago

Payment protocol phase 1

1. Debitor loads the billsEntry file from the DHT (after user identifies transaction).
   - Content copied to fields of ledgerPE1 and creditorFADebitorPE1 (see below).
2. Debitor stores the following ephemeral files in the DHT.
   - LedgerPE1.
     First 11 fields of an AccountLedger entry (contains billsEntry_DHTHash).
   - CreditorFADebitorPE1.
     First 6 fields of a FAM entry.
3. Debitor notifies creditor of availability of above two files.
   - Ephemeral file info sent in form of two ordered pairs: (fileID, DHT-Hash).
4. Creditor loads the above two files and then the billsEntry file from the DHT.
   - Checks fields of ephemeral files are correct; in particular, checks that:
     - LedgerPE1 fields correspond to billsEntry fields.

Payment protocol phase 2

1. Creditor stores the following ephemeral files in the DHT:
   - LedgerPE2.
     LedgerPE1 fields + next 4 entry fields including creditor signature.
   - CreditorFADebitorPE2.
     CreditorFADebitorPE1 fields + next 5 entry fields including creditor signature.

- LedgerPE1 (no need to store a new file, simply notify same phase 1 file).
  First 11 fields of an AccountLedger entry.
- DebitorFACreditorPE1.
  First 6 fields of a FAM entry

and the following persistent file in the DHT:

- FBMEntry.
  Creditor FBM entry, signed by creditor, the author of the feedback note that FBMEntry . otherFAMEntry_DHTHash = CreditorFA-DebitorPE1 . self_previous_FAMEntry_DHTHash . self_next_FAMEntry_DHTHash

2. Creditor notifies debitor of the availability of the above five files.
   - Sending the information in the form of five ordered pairs: (fileID, DHTHash).
3. Debitor loads the above five files from the DHT.
   - Checks fields of ephemeral files are correct; in particular, checks that:
     - LedgerPE1 fields correspond to billsEntry fields.
     - Prefix(11, ledgerPE2) = ledgerPE1.
     - Prefix(6, creditorFADebitorPE2) = creditorFADebitorPE1.
     - LedgerPE2.previous_ledgerEntry_DHTHash correct (and links back).
     - CreditorFADebitorPE2.previous_FAMEntry_DHTHash correct (+ link).
     - Creditor digital signatures in ledgerPE2 & creditorFADebitorPE2 correct.
   - Checks fields of persistent files are correct; in particular, checks that:
     - Creditor FBMEntry and creditorFADebitorPE2 contain same feedback.
     - FBMEntry.previous_FBMEntry_DHTHash correct (and links back).

- Creditor digital signature correct.

Payment protocol phase 3

1. Debitor stores the following ephemeral files in the DHT:

   - LedgerPE3.
     LedgerPE1 data + next 4 entry fields including debitor signature.
   - DebitorFACreditorPE2.
     DebitorFACreditorPE1 data + next 5 entry fields including debitor signature.

   and the following persistent files in the DHT:

   - FBMEntry.
     Debitor FBM entry, signed by debitor, the author of the feedback note that FBMEntry . other_FAMEntry_DHTHash = CreditorFA-DebitorPE1 . self_previous_FAMEntry_DHTHash . self_next_FAMEntry_DHTHash.
   - LedgerEntry.
     Debitor AccountLedger entry, signed by creditor, then by debitor.
   - FAMEntry.
     Debitor FAM entry, signed by creditor, author of the feedback, then by debitor.

2. Debitor notifies creditor of the availability of the above five files.

   - Sending the information in the form of five ordered pairs: (fileID, DHTHash).

3. Creditor loads the above five files from the DHT.

   - Checks fields of ephemeral files are correct; in particular, checks that:
     - Prefix(11, ledgerPE3) = ledgerPE1.
     - Prefix(6, debitorFACreditorPE2) = debitorFACreditorPE1.
     - LedgerPE3 . previous_ledgerEntry_DHTHash correct (and links back).
     - DebitorFACreditorPE2.previous_FAMEntry_DHTHash correct (+ link).

- Debitor digital signatures in both ephemeral files correct.

- Checks fields of persistent files are correct; in particular, checks that:
  - Debitor FBMEntry and debitorFACreditorPE2 contain same feedback.
  - FBMEntry . previous_FBMEntry_DHTHash correct (and links back).
  - Prefix(11, FAMEntry) = creditorFADebitorPE2.
  - Prefix(18, ledgerEntry = ledgerPE2.
  - Debitor digital signature in all three persistent files correct.

Payment protocol phase 4

1. Creditor stores the following persistent files in the DHT.
   - LedgerEntry.
     Creditor AccountLedger entry, signed by debitor, then by creditor.
   - FAMEntry.
     Creditor FAMEntry signed by debitor, author of the feedback, then by creditor.
2. Creditor notifies debitor of the availability of the above two files.
   - Sending the information in the form of two ordered pairs: (fileID, DHTHash).
3. Debitor loads the above two files from the DHT.
   - Checks fields of persistent files are correct; in particular, checks that:
     - Prefix(11, FAMEntry) = debitorFACreditorPE2.
     - Prefix(18, ledgerEntry) = ledgerPE3.
     - LedgerEntry_creditor . other_ledgerEntry_DHTHash = ledgerEntry_debitor_DHTHash.
     - Creditor digital signature in both persistent files correct.

Possible checks:
File persistent checks conducted by creditor

- LedgerEntry_debitor . other_ledgerEntry_DHTHash = ledgerEntry_creditor_DHTHash.
- LedgerEntry_debitor . self_FBMEntry_DHTHash = FBMEntry_debitor_DHTHash.
- LedgerEntry_debitor . other_FBMEntry_DHTHash = FBMEntry_creditor_DHTHash.
- FBMEntry_debiitor . other_FAMEntry_DHTHash = FAMEntry_creditor_DHTHash.
- FAMEntry_debitor . other_FBMEntry_DHTHash = FBMEntry_creditor_DHTHash.

File persistent checks conducted by debitor

- LedgerEntry_creditor . self_FBMEntry_DHTHash = FBMEntry_creditor_DHTHash.
- LedgerEntry_creditor . other_FBMEntry_DHTHash = FBMEntry_debitor_DHTHash.
- FBMEntry_creditor . other_FAMEntry_DHTHash = FAMEntry_debitor_DHTHash.
- FAMEntry_creditor . other_FBMEntry_DHTHash = FBMEntry_debitor_DHTHash.

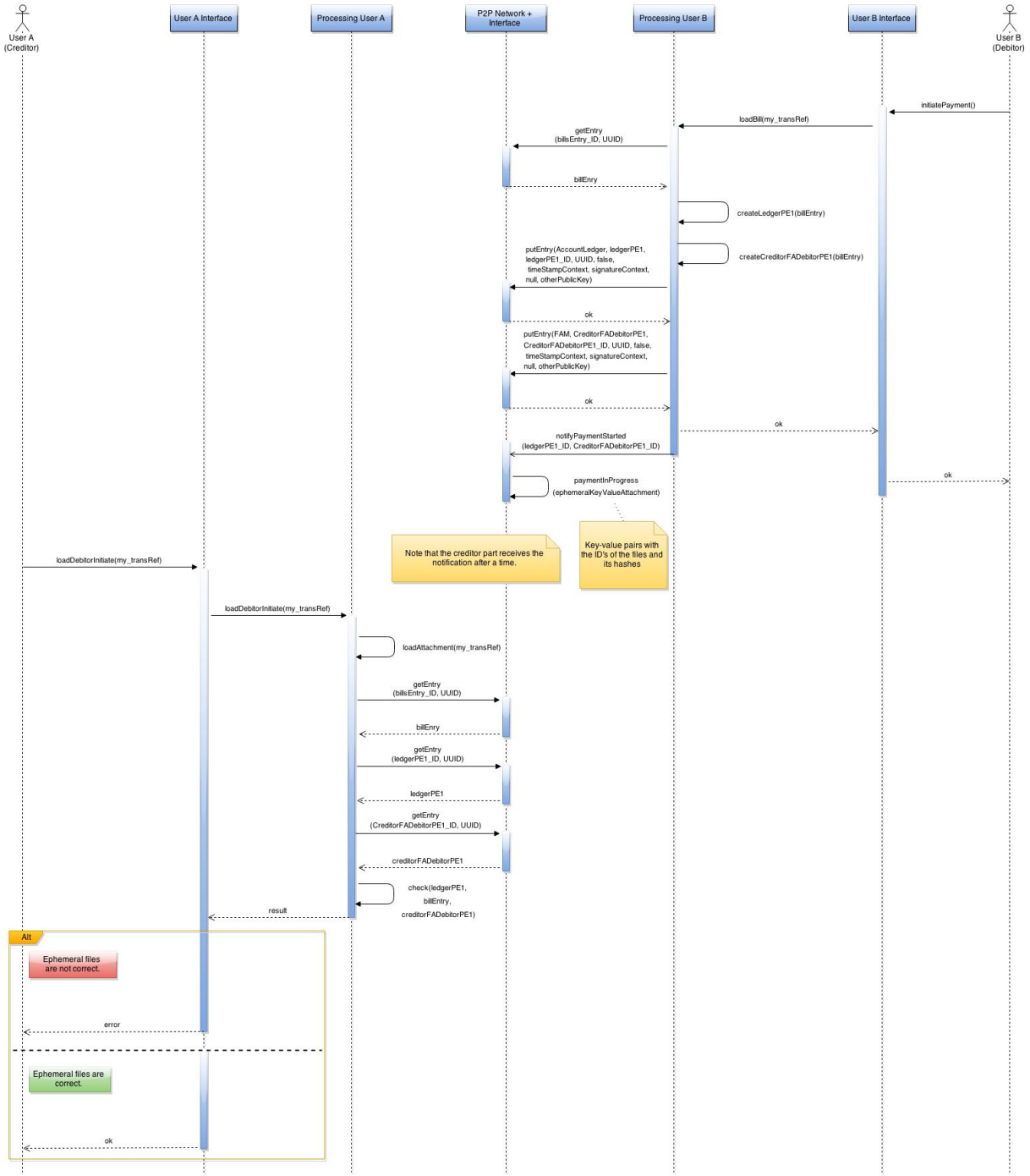

Figura B.15: Diagrama de secuencia de efectuar pago (parte 1).

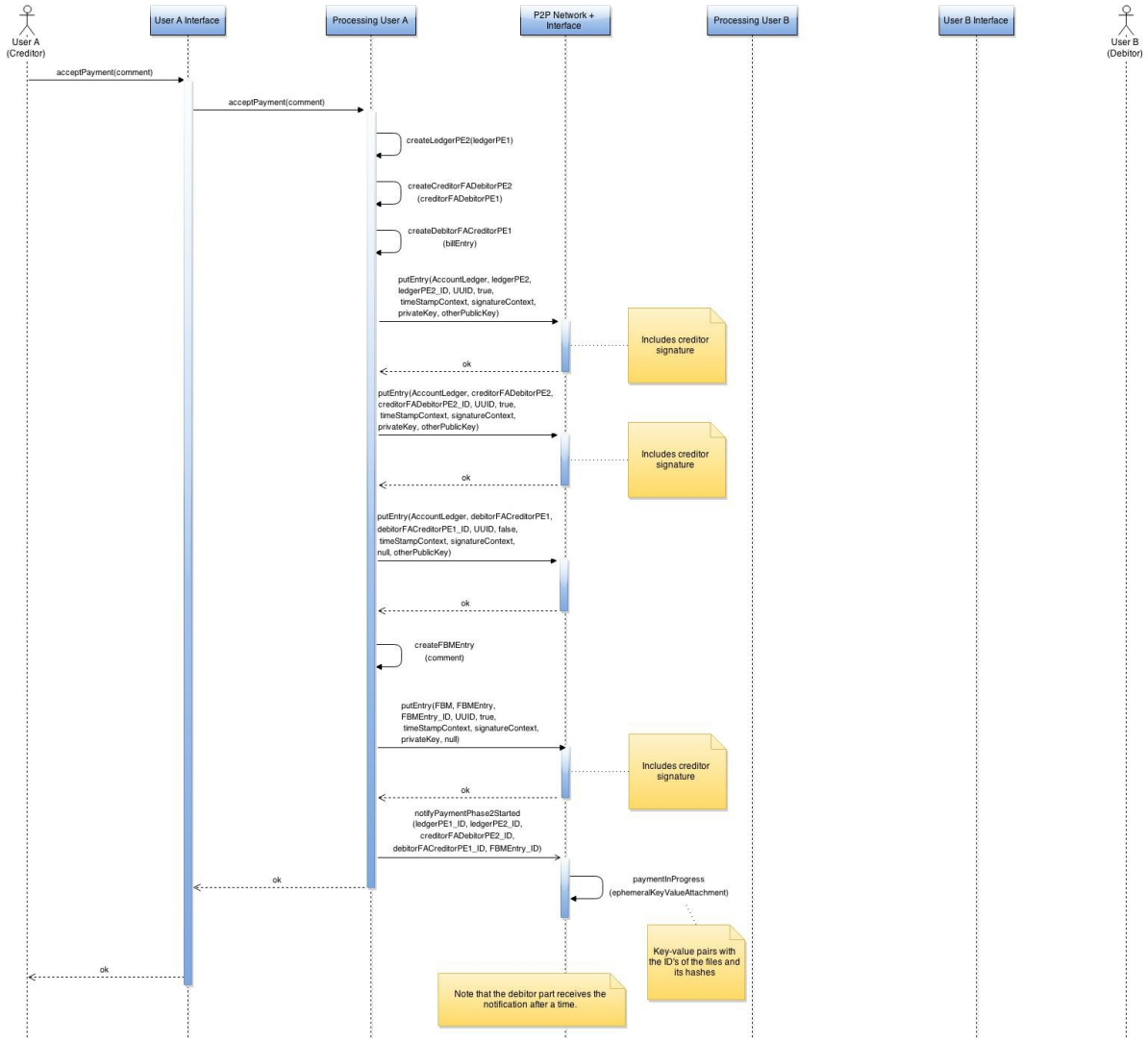

Figura B.16: Diagrama de secuencia de efectuar pago (parte 2).

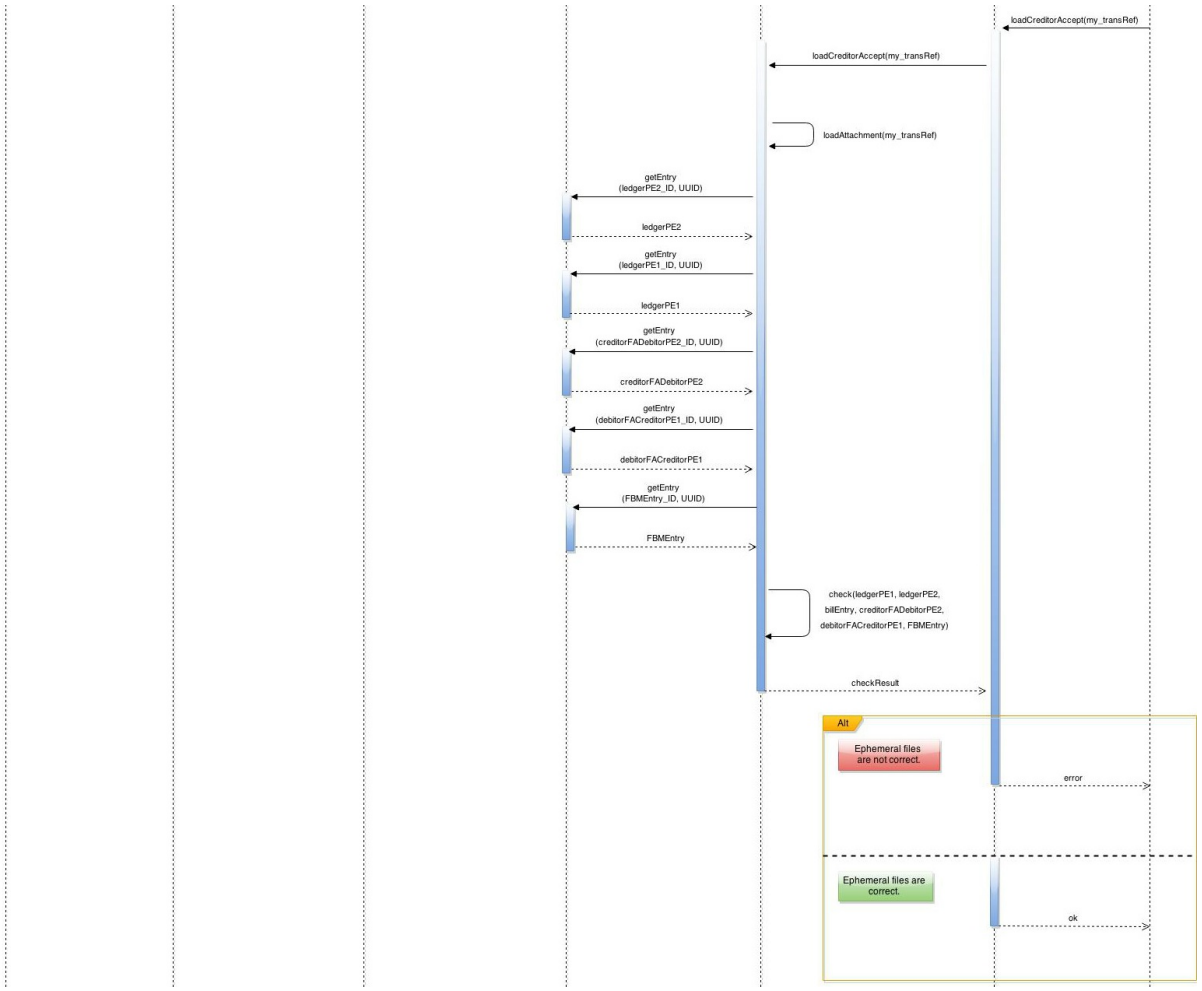

Figura B.17: Diagrama de secuencia de efectuar pago (parte 2).

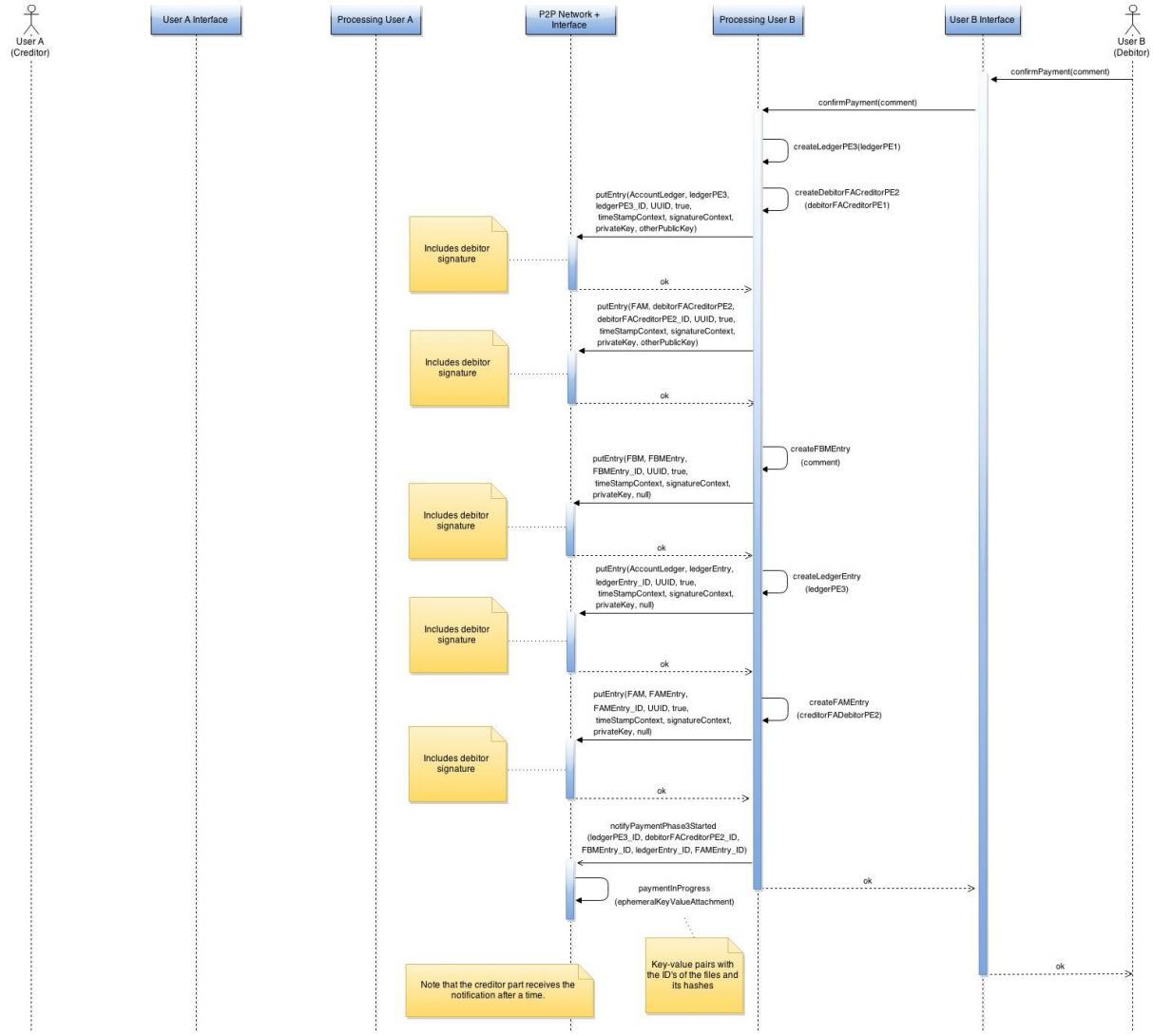

Figura B.18: Diagrama de secuencia de efectuar pago (parte 3).

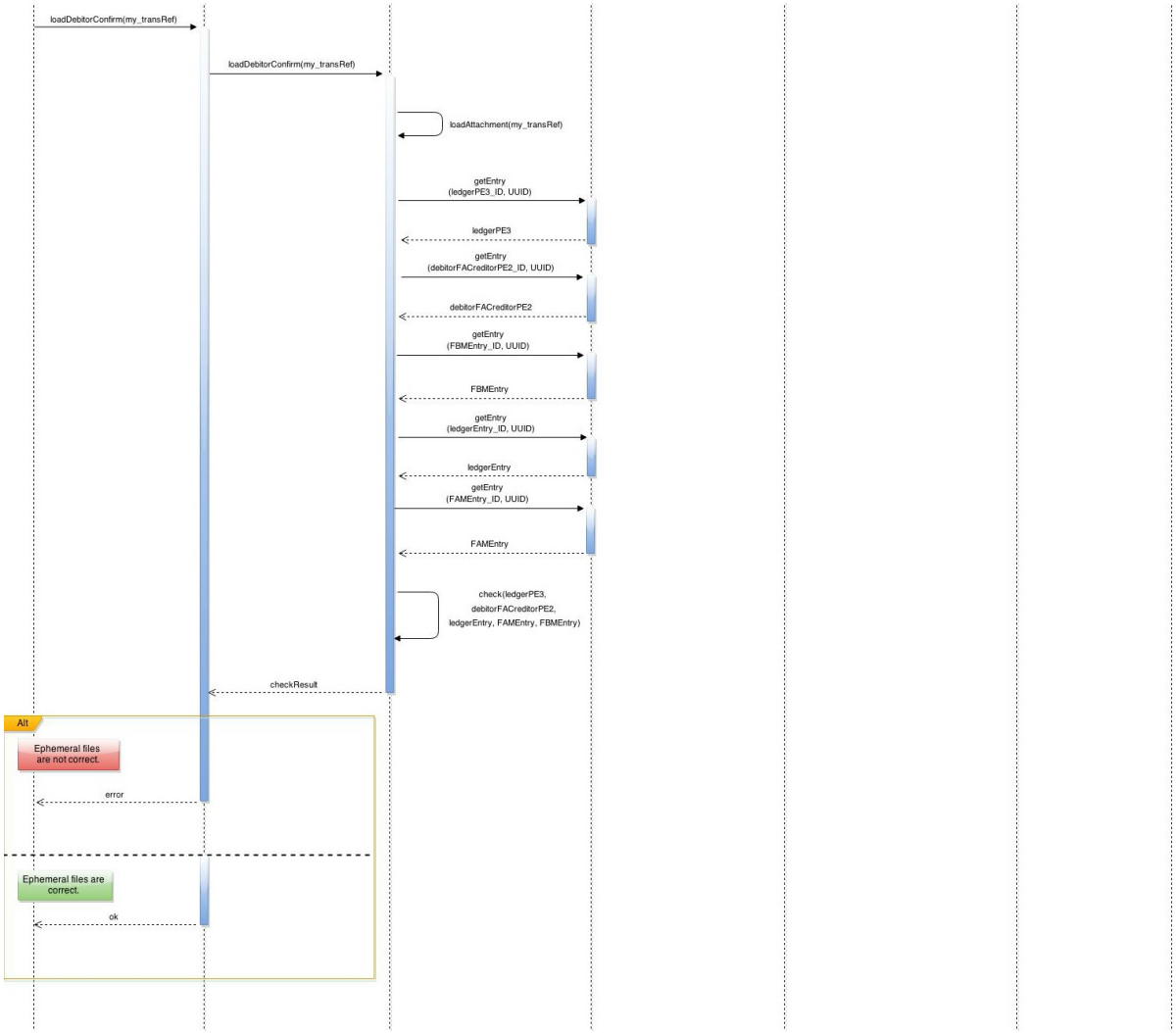

Figura B.19: Diagrama de secuencia de efectuar pago (parte 3).

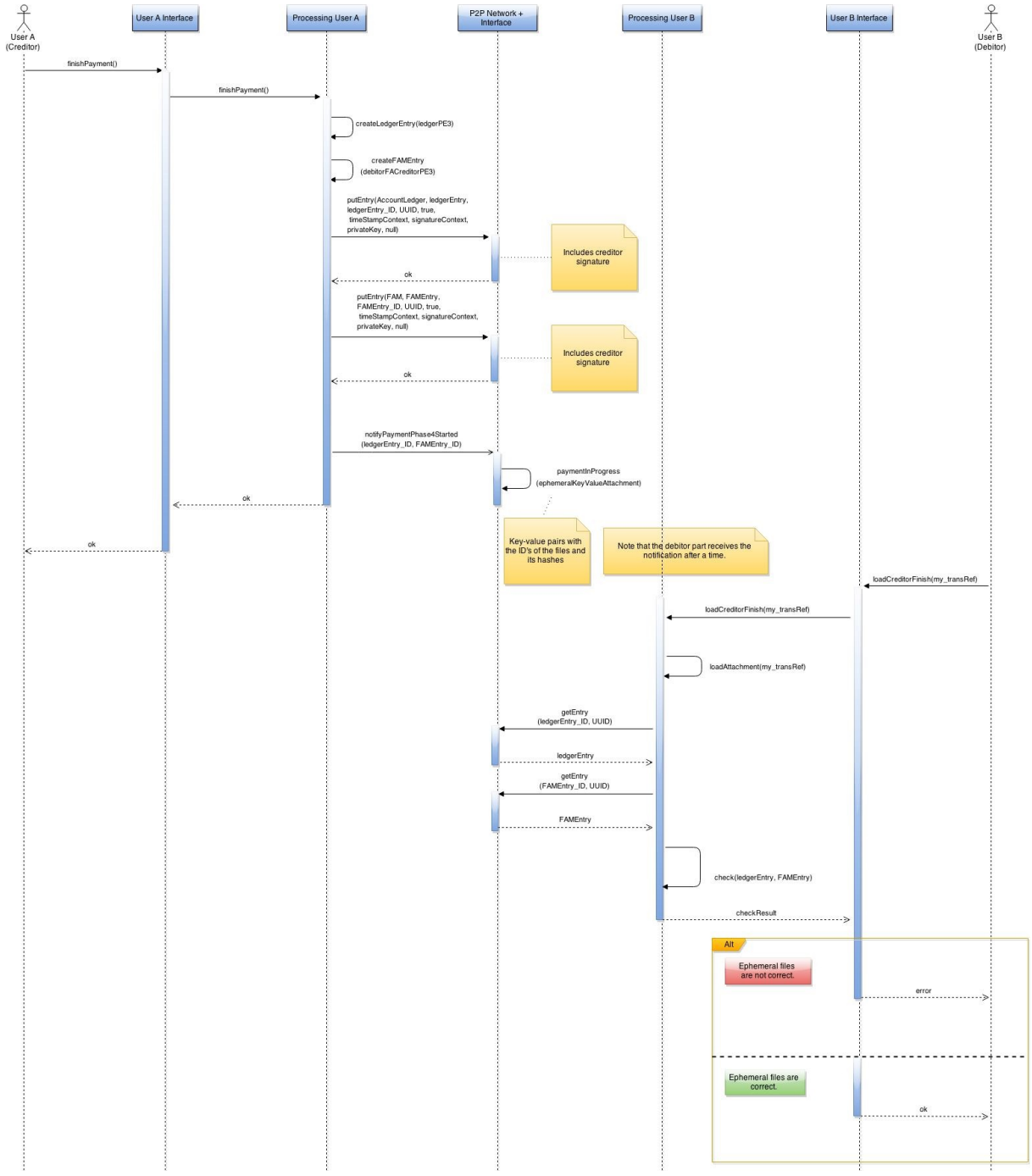

Figura B.20: Diagrama de secuencia de efectuar pago (parte 4).

Negociación de presupuesto / factura proforma / factura

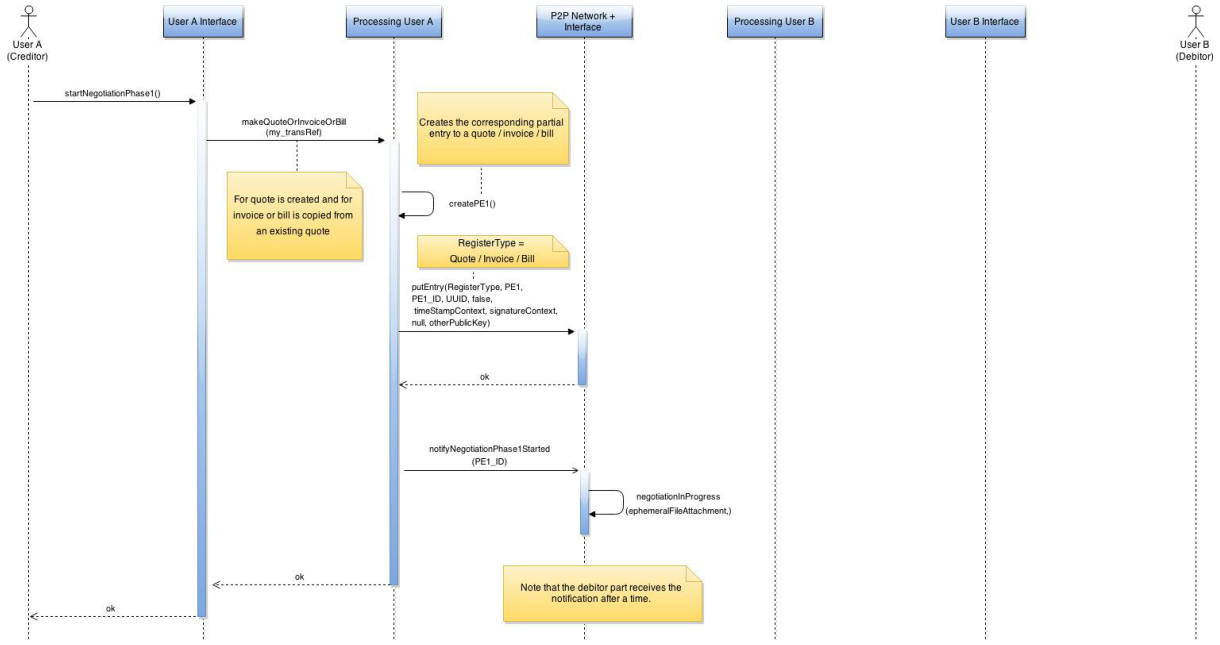

Figura B.21: Diagrama de secuencia de iniciar negociación de presupuesto / factura proforma / factura (fase 1).

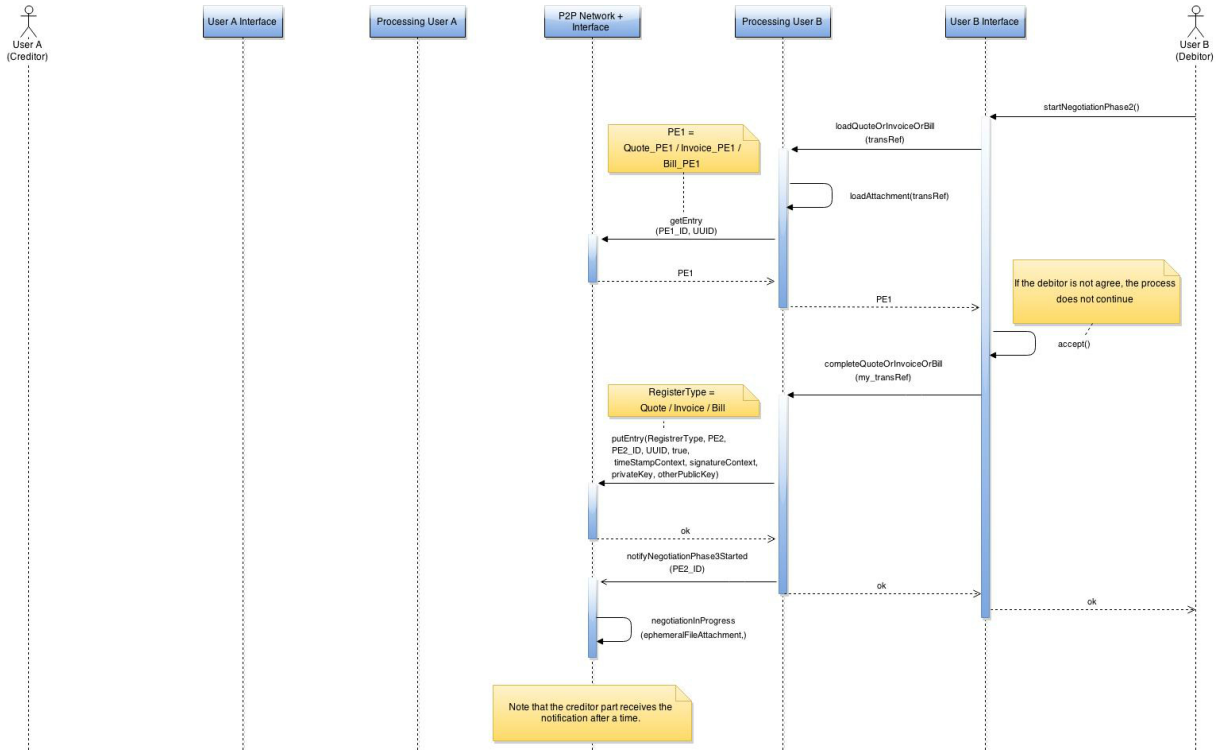

Figura B.22: Diagrama de secuencia de iniciar negociación de presupuesto / factura proforma / factura (fase 2).

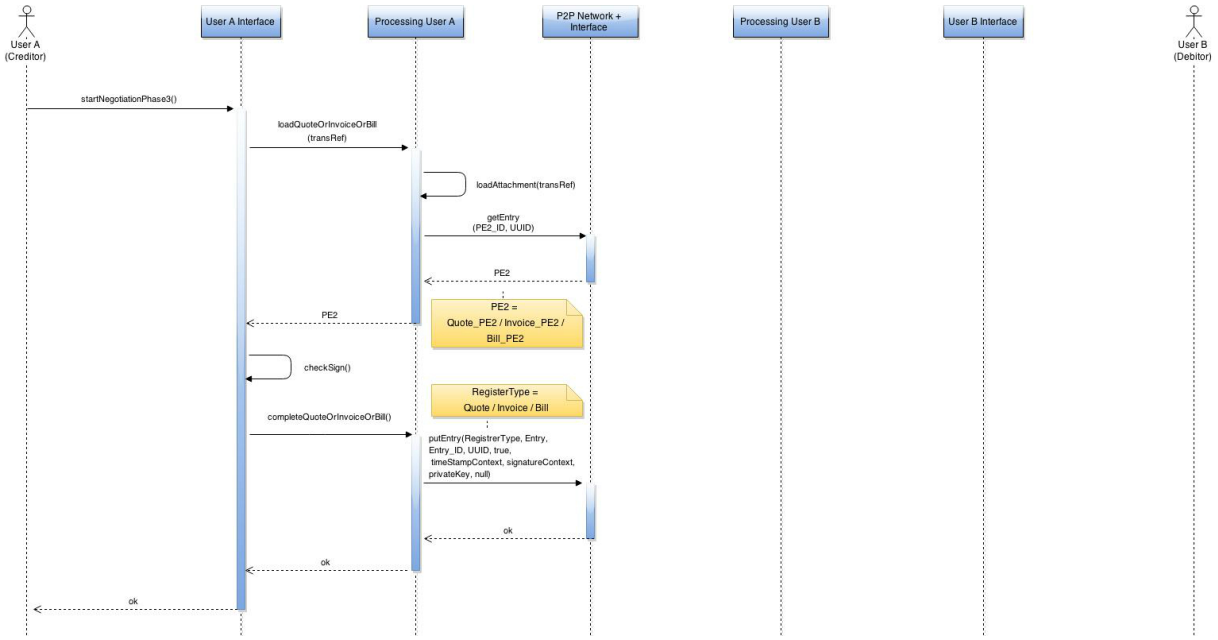

Figura B.23: Diagrama de secuencia de iniciar negociación de presupuesto / factura proforma / factura (fase 3).

Apéndice C

Documentos de diseño

C.1. Diagramas de clases

Los diagramas de clases que se muestran a continuación muestran una visión actual de la implementación. Además permiten observar los patrones de diseño que se comentaron en el capítulo de implementación. Hay que tener en cuenta que en prácticamente todos los diagramas se han ocultado algunas asociaciones con roles, multiplicidades y más indicaciones con el objetivo de ayudar a la visibilidad de los mismos (demasiadas líneas crearían confusión).

C.1.1. Estructura global de paquetes

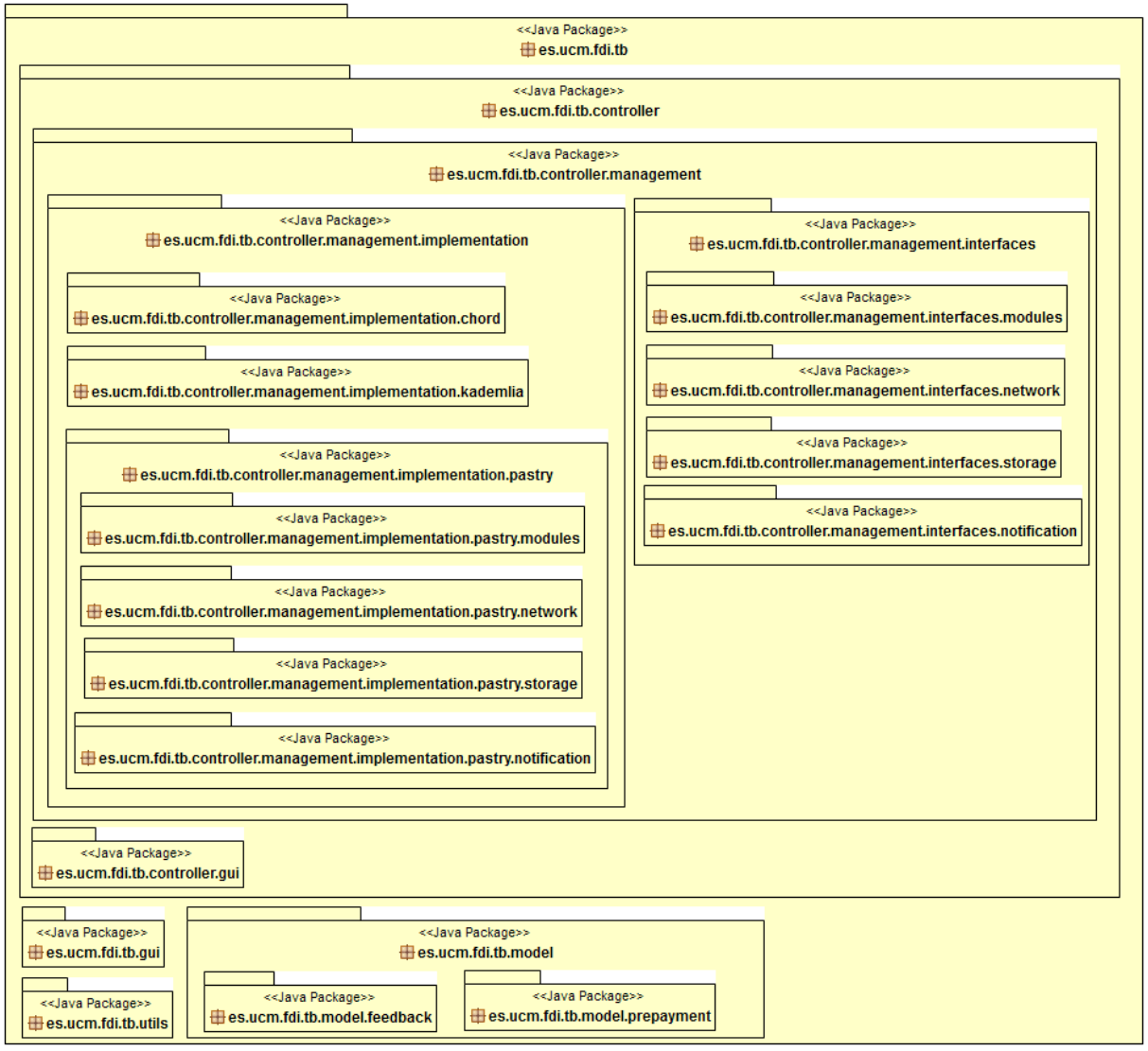

Figura C.1: Diagrama de clases con la estructura de todos los paquetes.

C.1.2. Modelo

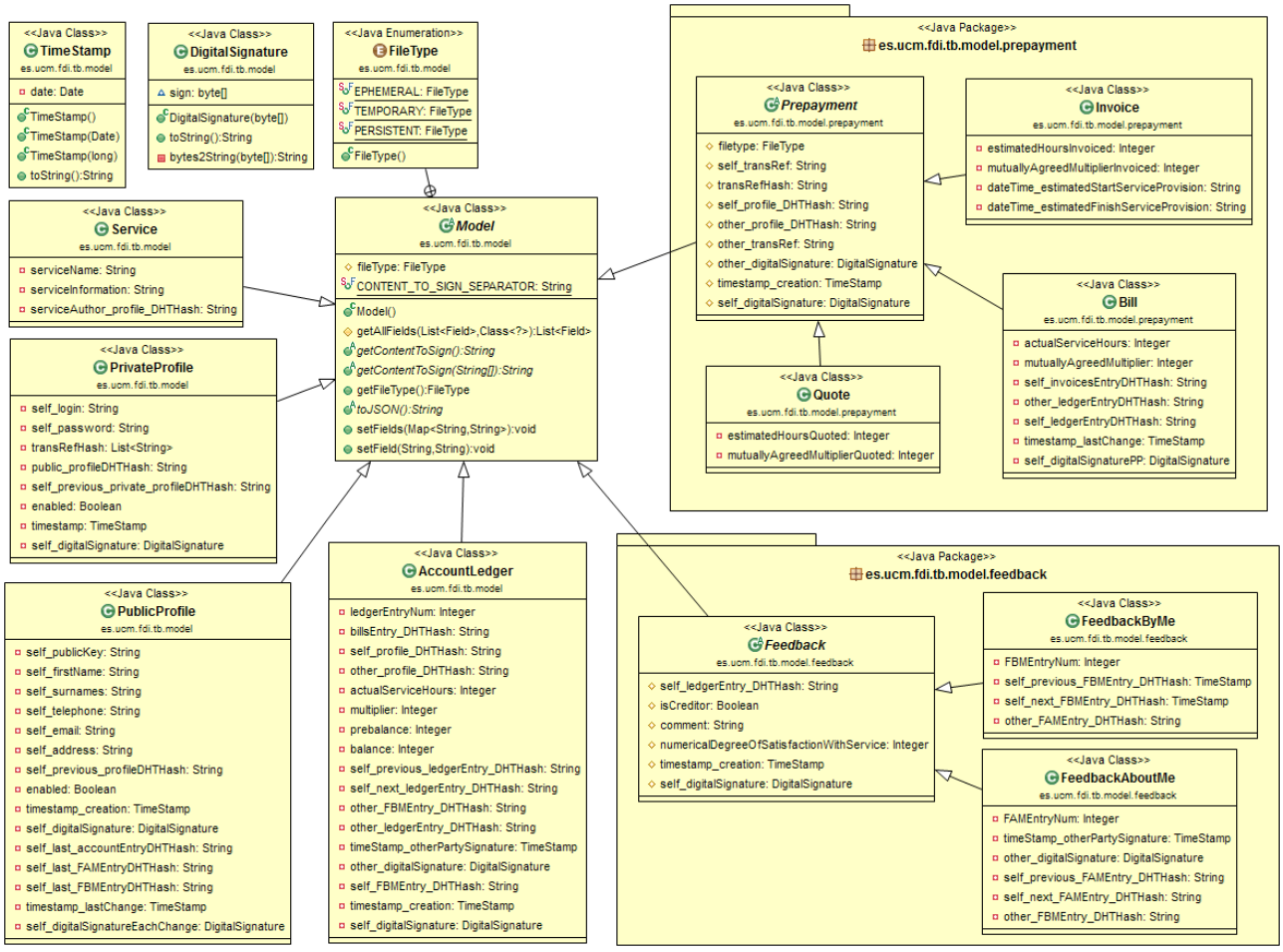

Figura C.2: Diagrama de clases del modelo.

C.1.3. Vista

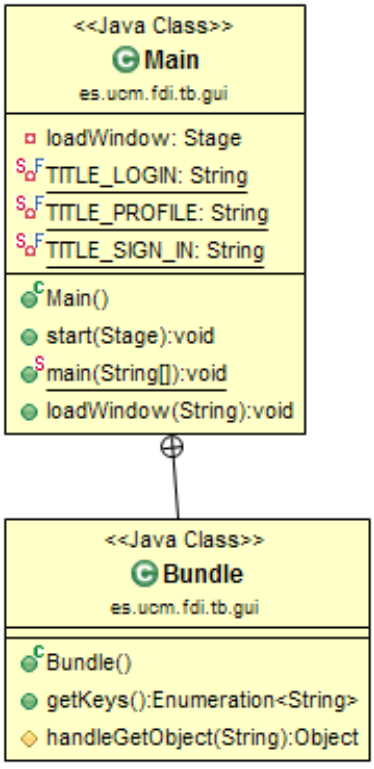

Figura C.3: Diagrama de clases de la vista.

C.1.4. Controlador

Interfaces

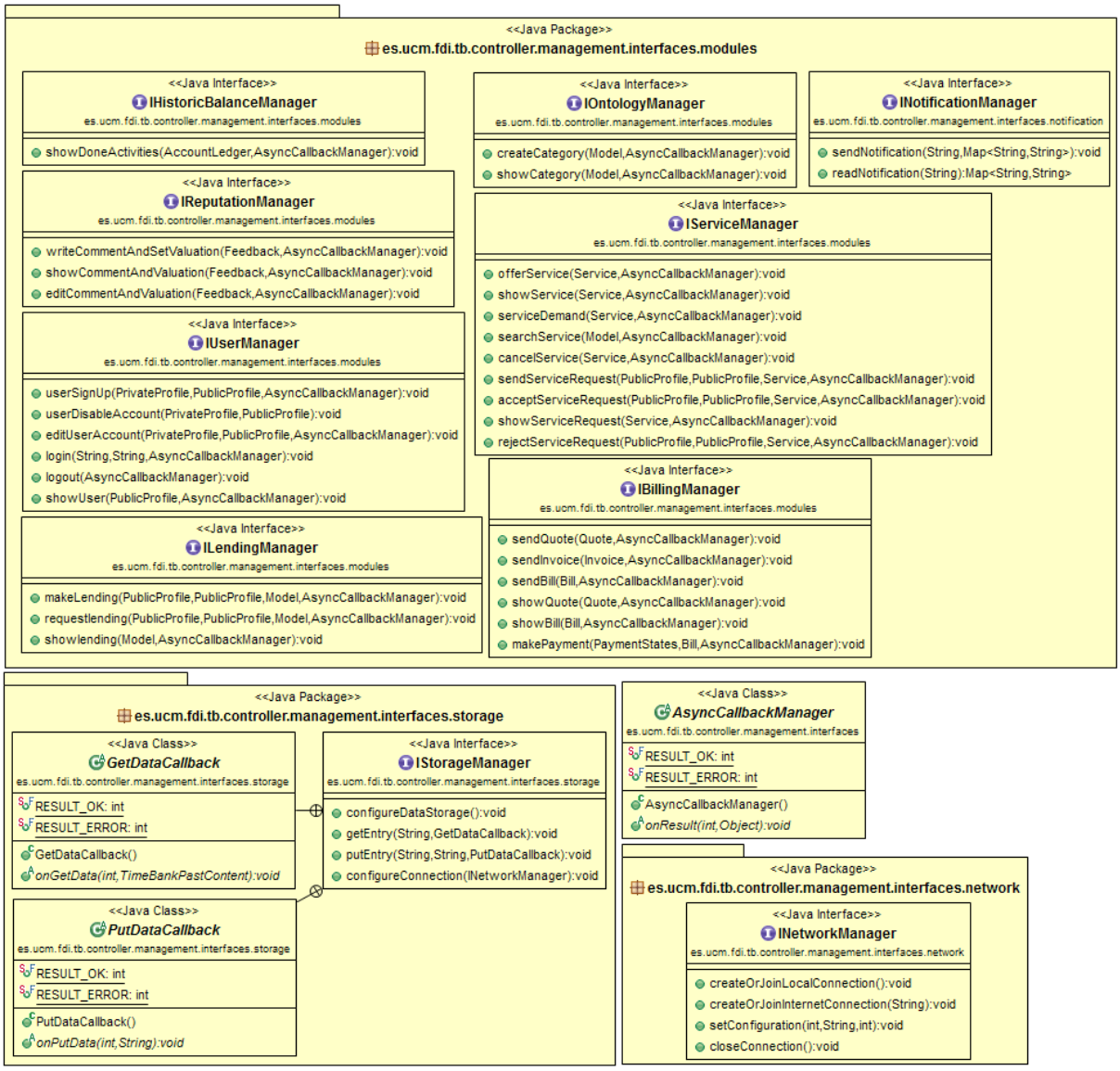

Figura C.4: Diagrama de clases de las interfaces del controlador (1).

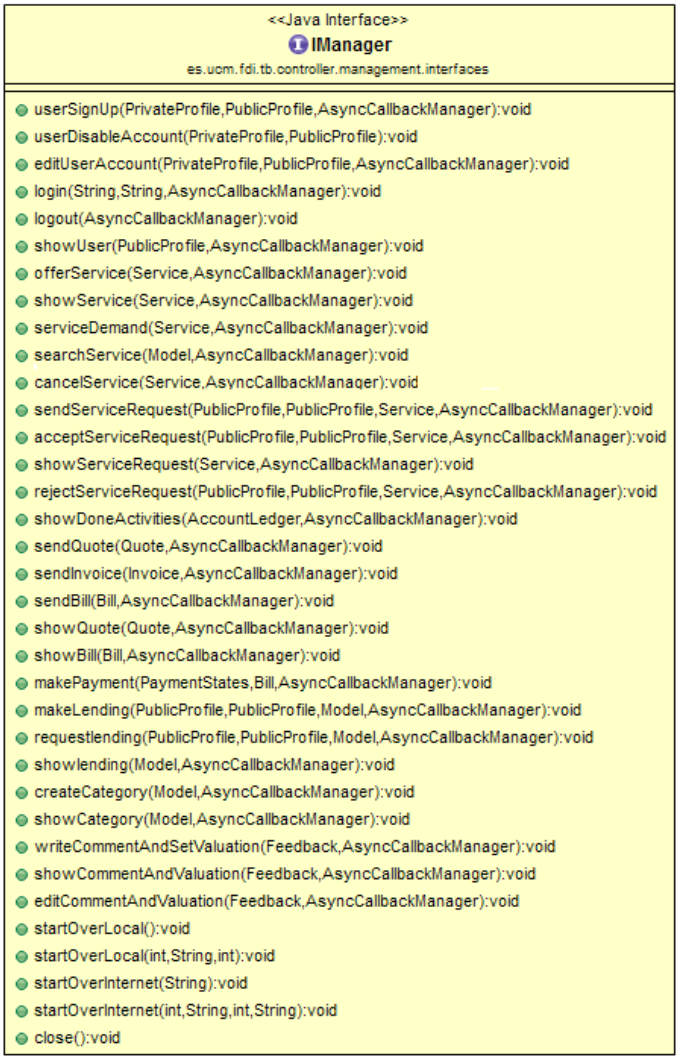

Figura C.5: Diagrama de clases de las interfaces del controlador (2).

Implementación

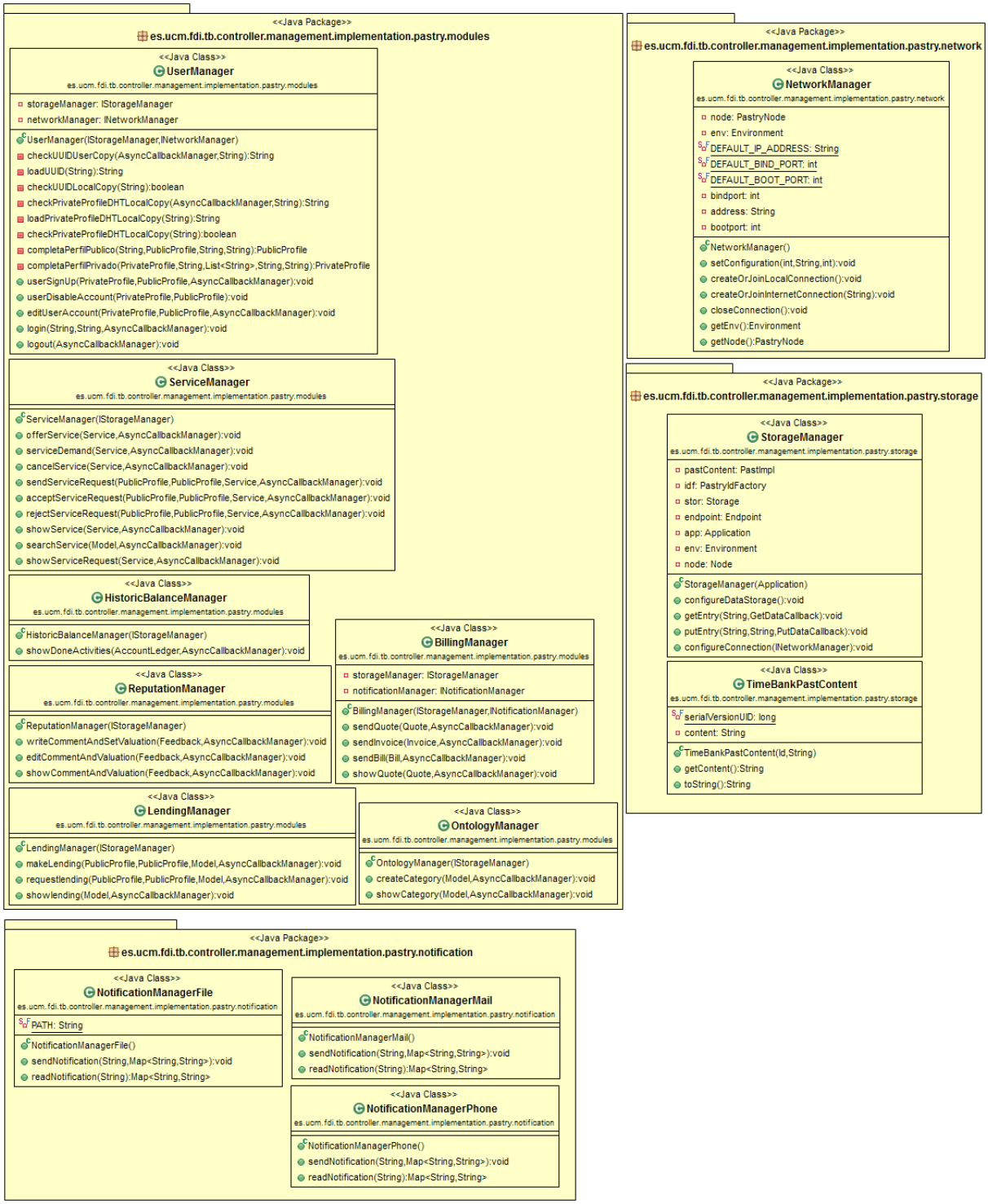

Figura C.6: Diagrama de clases de las implementaciones del controlador (1).

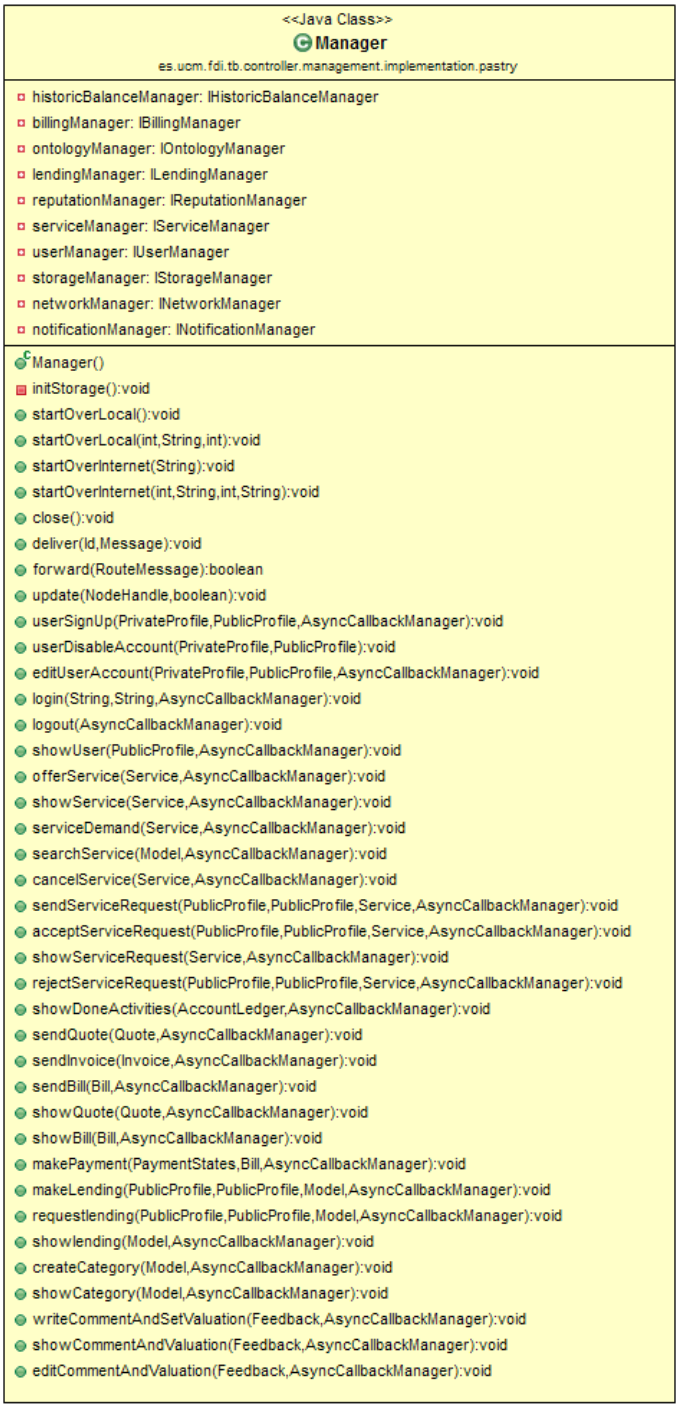

Figura C.7: Diagrama de clases de las implementaciones del controlador (2).

C.1.5. Útiles

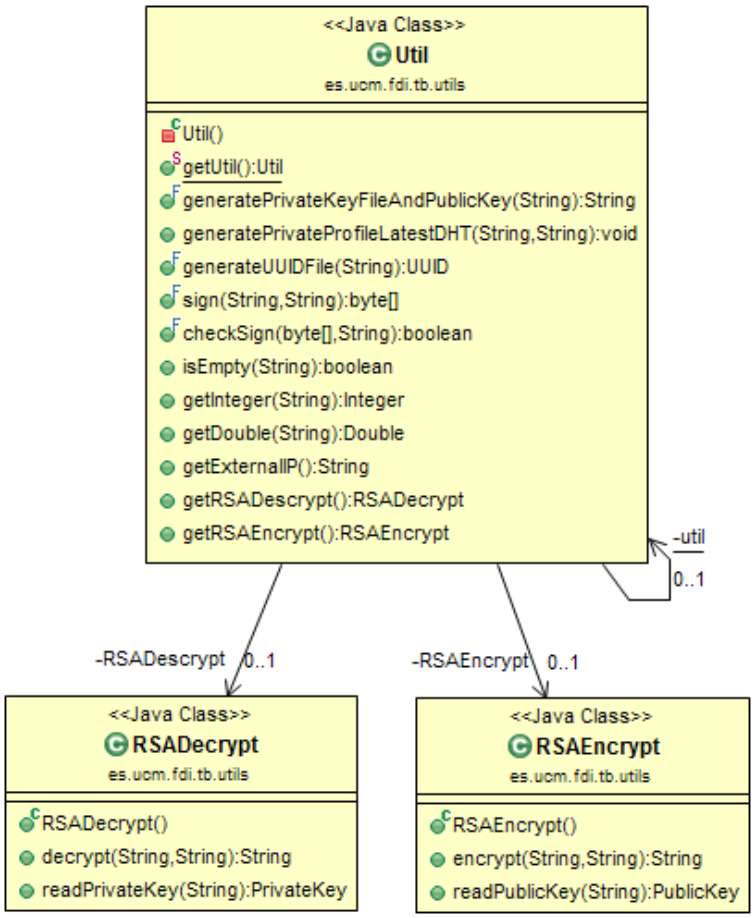

Figura C.8: Diagrama de clases de las utilidades.

Apéndice D

Ficheros del sistema

D.1. Ficheros del sistema en tablas

We assume the following:

- Files cannot be deleted.
- The filetype is one of the following. If the underlying system does not provide this functionality, each node can apply it to the files it stores.
    - Ephemeral: file expires if it remains unchanged for x hours. Ephemeral files would generally be encrypted (for viewing by other party only).
    - Temporary: file expires if it remains unchanged for 1 month. Content (i.e. not timestamp and signature) could be encrypted (for viewing by other party only). To avoid a persistent file that has been unchanged for 5 years expiring, timestamp and sign it again.
    - Persistent: file expires if it remains unchanged for 5 years. Financial data should be available for 5 years, e.g. for tax reasons in Belgium.
- Non-obligatory fields could either have a default value or not be present. Examples of default value: 0, empty_string, as_in_previous_phase.
- UUIDs (Universally Unique Identifiers) are based on RFC 4122. For security reasons, UUIDs should not be made public. This fact greatly affects the design.

- Timestamps could use OriginStamp
- In filenames, “my” refers to the local-file owner (always local party);
- In fields constituting the file contents, “self” refers to the DHT-file owner [78] [30]

Private Profile	Persistent file. One file per user. DHT-file owner is user.
Mutable fields in italics. The profile changes with each transaction. privateProfile_ID = privateProfileString privateProfile_DHTHash = makeDHTHash(self_UUID, privateProfileString)	
filetype	Value = persistent.
self_login	Login
self_password	Encrypted (but still risky!!).
list(TransRefHash)	LIST OF AS-YET UNFINISHED TRANSACTIONS FOR USER
public_profileDHTHash	Updated if new public profile created
self_previous_private_profileDHTHash	Empty string if no previous; any change to field 3: new profile created, oldProfile.enabled = false
enabled	Initialised to true. Updated if user leaves the system or if new private profile created (old private profiles not deleted)
self_digitalSignature	User signs this on creating it

Cuadro D.1: Private Profile file

Public Profile	Persistent file. One file per user. DHT-file owner is user.
Mutable fields in italics. The profile changes with each completed payment. publicProfile_ID = publicProfileString publicProfile_DHTHash = makeDHTHash(self_UUID, publicProfileString)	
filetype	Value = persistent.
self_publicKey	self_UUID and self_privateKey are not made public but MUST be backed up
self_firstName	Not obligatory (since will not be verified!)
self_surnames	Not obligatory (since will not be verified!)
self_telephone	One of phone or e-mail is needed for notifications and external negotiation; e-mail preferred
self_e-mail	One of phone or e-mail is needed for notifications and external negotiation; e-mail preferred
self_address	Not obligatory (since will not be verified!)
self_previous_profileDHTHash	Empty string if no previous; any change to fields 2-8: new profile created, oldProfile.enabled = false
enabled	Initialised to true. Updated if user leaves the system or if new profile created (old profiles not deleted)
timestamp_creation	When this profile was created
self_digitalSignature_creation	File owner signs (some concrete representation of) the preceding fields on creating this profile
self_last_LedgerEntryDHTHash	Updated in each PP in which this user participates. Can obtain current balance & last activity.
self_last_FAMEntryDHTHash	Updated in each PP in which this user participates.
self_last_FBMEntryDHTHash	Updated in each PP in which this user participates.
timestamp_lastChange	Added just before each signing of the profile (during last participation in PP).
self_digitalSignatureLastChange	File owner signs this profile after each change (during each participation in PP)

Cuadro D.2: Public Profile file

Quotes/Estimates (presupuestos)	Temporary file. One file per transaction. DHT-file owner is creditor.
No mutable fields. quotesEntry_ID = concat(quotesString, my_transRef); each party uses different local filename quotesEntry_DHTHash = makeDHTHash(creditor_UUID, creditor_quotesEntry_ID)	
filetype	Value = temporary
self_transRef	Locally unique (to the creditor) reference chosen by creditor to refer to this transaction
transRefHash	makeDHTHash(self_UUID, self_transRef); self = creditor
self_profile_DHTHash	Creditor's public profile; debitor has already obtained the creditor's public key in the service offer-demand process
other_profile_DHTHash	Debitor's public profile; creditor has already obtained the debitor's public key and profile hash in the offer-demand process
estimatedHours quoted	
mutuallyAgreedMultiplier quoted	If the system permits the use of a multiplier
other_transRef	Locally unique (to the debitor) reference chosen by debitor to refer to this transaction
other_digitalSignature	Debitor signs (some concrete representation of) the preceding fields (via ephemeral file)
timestamp_creation	Added just before the signing of this quote
self_digitalSignature	Creditor signs this quote on creating it
Two ephemeral files, partial entries (PE), used in the Quote Negotiation Protocol (QNP)	
* quotesPE1_ID = concat(quoteString, transRefHash, "p1")	Created by creditor in QNP phase 1, notified to debitor; last 4 fields are empty
quotesPE1_DHTHash = makeDHTHash(creditor_UUID, first 7 fields)	DHT Hash created from creditor_UUID and (some concrete representation of) the first 7 fields
* quotesPE2_ID = concat(quotesString, transRefHash, "p2")	Created by debitor in QNP phase 2, notified to creditor; last 2 fields are empty
quotesPE2_DHTHash = makeDHTHash(debitor_UUID, first 9 fields)	DHT Hash created from debitor_UUID and (some concrete representation of) the first 9 fields

Cuadro D.3: Quotes file

Invoices (facturas proforma)	Persistent file. One file per transaction. DHT-file owner is creditor.
No mutable fields. invoicesEntry_ID = concat(invoicesString, my_transRef); each party uses different local filename invoicesEntry_DHTHash = makeDHTHash(creditor_UUID, creditor_invoicesEntry_ID)	
filetype	Value = persistent.
self_transRef	Copied from the corresponding quote
other_transRef	Copied from the corresponding quote
transRefHash	Copied from the corresponding quote
self_profile_DHTHash	Copied from the corresponding quote
other_profile_DHTHash	Copied from the corresponding quote
estimatedHours invoiced	By default, same value as in quote, but different value is possible
mutuallyAgreedMultiplier invoiced	By default, same value as in quote, but different value is possible
dateTime_estimatedStartServiceProvision	
dateTime_estimatedFinishServiceProvision	Not obligatory
other_digitalSignature	Debitor signs (some concrete representation of) the preceding fields (via ephemeral file)
timestamp_creation	Added just before the signing of this invoice
self_digitalSignature	Creditor signs this invoice on creating it
Two ephemeral files, partial entries (PE), used in the Invoice Negotiation Protocol (INP)	
* invoicesPE1_ID = concat(invoiceString, creditor_transRefHash, "p1")	Created by creditor in INP phase 1, notified to debitor; last 3 fields are empty
invoicesPE1_DHTHash = makeDHTHash(creditor_UUID, first 10 fields)	DHT Hash created from (some concrete representation of) the first 10 fields
* invoicesPE2_ID = concat(invoicesString, creditor_transRefHash, "p2")	Created by debitor in INP phase 2, notified to creditor; last 2 fields are empty
invoicesPE2_DHTHash = makeDHTHash(debitor_UUID, first 11 fields)	DHT Hash created from (some concrete representation of) the first 11 fields

Cuadro D.4: Invoices file

Bills (facturas)	Persistent file. One file per transaction. File owner is creditor.
Mutable fields in italics. The bill changes once, during corresponding payment. billsEntry_ID = concat(billsString, my_transRef); each party uses different local filename billsEntry_DHTHash = makeDHTHash(creditor_UUID, billsEntry_ID)	
filetype	Value = persistent.
self_transRef	Copied from the corresponding invoice
other_transRef	Copied from the corresponding invoice
transRefHash	Copied from the corresponding invoice
self_profile_DHTHash	Copied from the corresponding invoice
other_profile_DHTHash	Copied from the corresponding invoice
mutuallyAgreedMultiplier	Copied from the corresponding invoice
self_invoicesEntryDHTHash	Corresponding invoice DHTHash
actualServiceHours	
other_digitalSignature	Debitor signs (some concrete representation of) the first six fields (via ephemeral file)
timestamp_creation	When this bill was created
self_digitalSignature_creation	Creditor signs (some concrete representation of) the preceding on creating this bill
other_ledgerEntryDHTHash	Corresponding entry in debitor's account ledger. Set during corresponding PP execution.
self_ledgerEntryDHTHash	Corresponding entry in creditor's account ledger. Set during corresponding PP execution.
timestamp_lastChange	Added just before signing of the complete bill (during corresponding PP execution)

Sigue en la página siguiente.

self_digitalSignature_lastChange	Creditor signs bill after last three fields added (during corresponding PP execution)
Two ephemeral files, partial entries (PE), used in the Bill Negotiation Protocol (BNP)	
" * billsPE1_ID = concat(billString, creditor_transRefHash, "p1")"	Created by creditor in BNP phase 1, notified to debitor; last 7 fields are empty
billsPE1_DHTHash = makeDHTHash(first 9 fields)	DHT Hash created from (some concrete representation of) the first 9 fields
" * billsPE2_ID = concat(billString, creditor_transRefHash, "p2")"	Created by debitor in BNP phase 2, notified to creditor; last 6 fields are empty
billsPE2_DHTHash = makeDHTHash(first 10 fields)	DHT Hash created from (some concrete representation of) the first 10 fields

Cuadro D.5: Bill file

AccountLedger	Persistent file. Two files per transaction. Owner of one DHT-file is creditor, owner of other is debitor.
No mutable fields. ledgerEntry_ID = concat(accountLedgerString, ledgerEntryNum) ledgerEntry_DHTHash = makeDHTHash(self_UUID, ledgerEntry_ID)	
filetype	Value = persistent.
ledgerEntryNum	Locally unique number chosen by the owner to refer to this account ledger entry
billsEntry_DHTHash	Corresponding bill
self_profile_DHTHash	Copied from bill. For debitor: ledgerEntry.self_profile_DHTHash = billsEntry.other_profile_DHTHash
other_profile_DHTHash	Copied from bill. For debitor: ledgerEntry.other_profile_DHTHash = billsEntry.self_profile_DHTHash
actualServiceHours	Copied from bill.
multiplier	Copied from bill.
pre-balance	Account balance (in hours) before this payment. Helps to avoid fraud.
balance	Account balance (in hours) after this payment
self_previous_ledgerEntry_DHTHash	makeDHTHash(self_UUID, self_previous_ledgerEntryNum). Cross-checked by other party.
self_next_ledgerEntry_DHTHash	makeDHTHash(self_UUID, self_next_ledgerEntryNum). Helps to avoid fraud. (**)

Sigue en la página siguiente.

other_FBMEntry_DHTHash	Feedback by other about my part in transaction, not obligatory
other_ledgerEntry_DHTHash	Same transaction in an entry of the account ledger of other party
timeStamp_otherPartySignature	Added by other party just before signing the preceding fields of this ledger entry
other_digitalSignature	Other party signs (some concrete representation of) the first fourteen fields
self_FBMEntry_DHTHash	Feedback by me about other's part in transaction, not obligatory.
timestamp_creation	Added just before the signing of this ledger entry
self_digitalSignature	File owner signs this ledger entry on creating it
Three ephemeral files, partial entries (PE), used in the Payment Protocol (PP)	
" * ledgerPE1_ID = concat(accountLedgerString,creditor_transRefHash, "p1")"	Created by debitor in PP phase 1, notified to creditor, renotified to debitor; last 7 fields are empty
ledgerPE1_DHTHash = makeDHTHash(first 11 fields)	DHT Hash created from (some concrete representation of) the first 11 fields
" * ledgerPE2_ID = concat(accountLedgerString,creditor_transRefHash, "p2")"	Created by creditor in PP phase 2, notified to debitor; last 3 fields are empty
ledgerPE2_DHTHash = makeDHTHash(first 15 fields)	DHT Hash created from (some concrete representation of) the first 15 fields

Sigue en la página siguiente.

" * ledgerPE3_ID = concat(accountLedgerString, creditor_transRefHash, "p3")"	Created by debitor in PP phase 3, notified to creditor; last 3 fields are empty
ledgerPE3_DHTHash = makeDHTHash(first 15 fields)	DHT Hash created from (some concrete representation of) the first 15 fields

Cuadro D.6: Account Ledger file

Feedback By Me (FBM)	Persistent file. One file per transaction feedback by a given party. DHT-file owner is subject of feedback.
No mutable fields. FBMEntry_ID = concat(FBMString, FBMEntryNum) FBMEntry_DHTHash = makeDHTHash(self_UUID, FBMEntry_ID)	
filetype	Value = persistent.
FBMEntryNum	Locally unique number chosen by the owner to refer to this FBM entry
self_ledgerEntry_DHTHash	Account ledger entry, with same owner as this FBM file, to which the feedback refers
isCreditor	Boolean indicating if this party was creditor or debitor in the transaction to which feedback refers
comment	
numericalDegreeOfSatisfactionWithService	
other_FAMEntry_DHTHash	Same feedback in a FAM entry belonging to the other party
self_previous_FBMEntry_DHTHash	makeDHTHash(self_UUID, self_previous_FBMEntryNum). To navigate through entries.
self_next_FBMEntry_DHTHash	makeDHTHash(self_UUID, self_next_FBMEntryNum). Helps to avoid fraud.
timestamp_creation	Added just before signing of this FBM entry
self_digitalSignature	File owner signs this FBM entry on creating it

Cuadro D.7: Feedback by Me file

Feedback About Me (FAM)	Persistent file. One file per transaction feedback about a given party. DHT-file owner is object of feedback.
No mutable fields. FAMEntry_ID = concat(FAMString, FAMEntryNum) FAMEntry_DHTHash = makeDHTHash(self_UUID, FAMEntry_ID)	
filetype	Value = persistent.
FAMEntryNum	Locally unique number chosen by the owner to refer to this FAM entry
self_ledgerEntry_DHTHash	Account ledger entry, with same owner as this FAM file, to which the feedback refers
isCreditor	Boolean indicating if this party was creditor or debitor in the transaction to which the feedback refers
self_previous_FAMEntry_DHTHash	makeDHTHash(self_UUID, self_previous_FAMEntryNum). To navigate through entries.
self_next_FAMEntry_DHTHash	makeDHTHash(self_UUID, self_next_FAMEntryNum). Helps to avoid fraud.
comment	
numericalDegreeOfSatisfactionWithService	
other_FBMEntry_DHTHash	Same feedback in a FBM entry belonging to the other party
timeStamp_otherPartySignature	Added by other party just before signing the preceding fields of this FAM entry
other_digitalSignature	Other party signs (some concrete representation of) the preceding fields + timestamp
timestamp_creation	Added just before signing of this FAM entry
self_digitalSignature	File owner signs this FAM entry on creating it

Sigue en la página siguiente.

Four ephemeral files, partial entries (PE), used in the Payment Protocol (PP)	
“ * creditorFADebitorPE1_ID = concat(FAMString, debitor_transRefHash, “p1”)”	Created by debitor in PP phase 1, notified to creditor; last 7 fields are empty
creditorFADebitorPE1_DHTHash = makeDHTHash(first 6 fields)	DHT Hash created from (some concrete representation of) the first 6 fields
“ * debitorFACreditorPE1_ID = concat(FAMString, creditor_transRefHash, “DCp2”)”	Created by creditor in PP phase 2, notified to creditor; last 7 fields are empty
debitorFACreditorPE1_DHTHash = makeDHTHash(first 6 fields)	DHT Hash created from (some concrete representation of) the first 6 fields
“ * creditorFADebitorPE2_ID = concat(FAMString, creditor_transRefHash, “CDp2”)”	Created by creditor in PP phase 2, notified to debitor; last 2 fields are empty
creditorFADebitorPE2_DHTHash = makeDHTHash(first 11 fields)	DHT Hash created from (some concrete representation of) the first 11 fields
“ * debitorFACreditorPE2_ID = concat(FAMString, debitor_transRefHash, “p3”)”	Created by debitor in PP phase 3, notified to creditor; last 2 fields are empty
debitorFACreditorPE2_DHTHash = makeDHTHash(first 11 fields)	DHT Hash created from (some concrete representation of) the first 11 fields

Cuadro D.8: Feedback about Me file

Local Files	
UUID	Generated in the signup
PrivateKey	Generated in the signup
PrivateProfileHash	privateProfile_DHTHash = makeDHTHash(self_UUID, privateProfile_ID)

Cuadro D.9: Explanation about local system files

(**) Having the other party sign the next_ledgerEntryDHTHash field gives some extra security at the cost of convenience. The convenience problem arises when a peer has several unfinished (i.e. concurrent) payment transactions. It has to use its value of next_localLedgerEntryNum / next_ledgerEntryDHTHash as the ledgerEntryNum / ledgerEntryDHTHash for the ledger entry of each of these unfinished transactions (the set of ledger entries must form a sequence, not a tree). The first payment to terminate gets to keep the assigned ledgerEntryNum / ledgerEntryDHTHash value (care needed here with the concurrency aspects!), while the other concurrent transactions have to abort and re-start with a different value for ledgerEntryNum / ledgerEntryDHTHash. However, this could be done by the software without bothering the external user.
The inclusion of the transRef in the files, instead of just the transRefHash, could lead to third parties deciphering the UUID from the study of many (transRef, transRefHash) pairs, but it means the user does not have to store the transRef value locally where it could easily be lost.

D.2. Ficheros del sistema en JSON

```
{
  "Private Profile": {
    "filetype": "persistent",
    "self_login": "...",
    "self_password": "...",
    "list(TransRefHash)": [],
    "public_profileDHTHash" : "...",
    "self_previous_private_profileDHTHash": "...",
    "enabled": true,
    "timestamp": "...",
    "self_digitalSignature_creation": "..."

  },

  "Public Profile" : {
    "filetype": "persistent",
    "self_publicKey": "...",
    "self_firstName": "...",
    "self_surnames": "...",
    "self_telephone": "...",
    "self_e-mail": "...",
    "self_address": "...",
    "self_previous_profileDHTHash": "...",
    "enabled": true,
    "timestamp_creation": "...",
    "self_digitalSignature_creation": "...",
    "self_last_LedgerEntryDHTHash": "...",
    "self_last_FAMEntryDHTHash": "...",
    "self_last_FBMEntryDHTHash": "...",
    "timestamp_lastChange": "...",
    "self_digitalSignatureLastChange": "..."
  },

  "Quotes/Estimates" : {
      "filetype": "temporary",
      "self_transRef": "...",
```

```
    "transRefHash": "...",
    "self_profile_DHTHash": "...",
    "other_profile_DHTHash": "...",
    "estimatedHours quoted": "...",
    "mutuallyAgreedMultiplier quoted": "...",
    "other_transRef": "...",
    "other_digitalSignature": "...",
    "timestamp_creation": "...",
    "self_digitalSignature": "..."
},

"Invoices" : {
    "filetype": "persistent",
    "self_transRef": "...",
    "other_transRef": "...",
    "transRefHash": "...",
    "self_profile_DHTHash": "...",
    "other_profile_DHTHash": "...",
    "estimatedHours invoiced": 0,
    "mutuallyAgreedMultiplier invoiced": 0,
    "dateTime_estimatedStartServiceProvision": "...",
    "dateTime_estimatedFinishServiceProvision": "...",
    "other_digitialSignature": "...",
    "timestamp_creation": "...",
    "self_digitalSignature": "..."
},

"Bills" : {
    "filetype": "persistent",
    "self_transRef": "...",
    "other_transRef": "...",
    "transRefHash": "...",
    "self_profileDHTHash": "...",
    "other_profileDHTHash": "...",
    "mutuallyAgreedMultiplier": 0,
    "self_invoicesEntryDHTHash": "...",
    "actualServiceHours": 0,
    "other_DigitialSignature": "...",
```

```
    "timestamp_creation": "...",
    "self_digitalSignature_creation": "...",
    "other_ledgerEntryDHTHash": "...",
    "self_ledgerEntryDHTHash": "...",
    "timestamp_lastChange": "...",
    "self_digitalSignature_lastChange": "..."
},

"AccountLedger" : {
    "filetype": "persistent",
    "ledgerEntryNum": 0,
    "billsEntry_DHTHash": "...",
    "self_profile_DHTHash": "...",
    "other_profile_DHTHash": "...",
    "actualServiceHours": 0,
    "multiplier": 0,
    "pre-balance": 0,
    "balance": 0,
    "self_previous_ledgerEntry_DHTHash": "...",
    "self_next_ledgerEntry_DHTHash": "...",
    "other_FBMEntry_DHTHash": "...",
    "other_ledgerEntry_DHTHash": "...",
    "timeStamp_otherPartySignature": "...",
    "other_digitalSignature": "...",
    "self_FBMEntry_DHTHash": "...",
    "timestamp_creation": "...",
    "self_digitalSignature": "..."
},

"FeedbackByMe" : {
    "filetype": "persistent",
    "FBMEntryNum": 0,
    "self_ledgerEntry_DHTHash": "...",
    "isCreditor": true,
    "comment": "...",
    "numericalDegreeOfSatisfactionWithService": 0,
    "other_FAMEntry_DHTHash": "...",
    "self_previous_FBMEntry_DHTHash": "...",
```

```
        "self_next_FBMEntry_DHTHash": "...",
        "timestamp_creation": "...",
        "self_digitalSignature": "..."
    },

    "FeedbackAboutMe" : {
        "filetype": "persistent",
        "FAMEntryNum": 0,
        "self_ledgerEntry_DHTHash": "...",
        "isCreditor": true,
        "self_previous_FAMEntry_DHTHash": "...",
        "self_next_FAMEntry_DHTHash": "...",
        "comment": "...",
        "numericalDegreeOfSatisfactionWithService": "...",
        "other_FBMEntry_DHTHash": "...",
        "timeStamp_otherPartySignature": "...",
        "other_digitalSignature": "...",
        "timestamp_creation": "...",
        "self_digitalSignature": "..."
    }

}
```

Apéndice E

Funciones primitivas

```
// Usage: entryID = getEntry(entryIDString). FieldValues is the type of lists of (fieldName, fieldValue) pairs.
FieldValues getEntry(entryIDString, String UUID){
        if (entryIDString == “privateProfile_ID”) {
                entryHash = makePersistentDHTHash(UUID, entryIDString)
        } else {
                entryHash = localLoad(entryIDString)
        }
        file newEntryFile = getDHT(entryHash)
        return newEntryFile.read()
}

// Append two fields -- timestamp and digital signature -- to entryData
// FieldValues is the type of lists of (fieldName, fieldValue) pairs.
FieldValues signEntry(FieldValues entryData, TimestampContext tContext, SignatureContext sContext, String
myPrivateKey){
        FieldValues newValues = []
        // Check whether should sign just the entryData, the entryData and the timestamp, or just the timestamp
        // The pseudocode here just signs the entryData
        newValues = append( newPair( makeSignatureFieldname(sContext), signature(entryData, myPrivatKey),
                            append( newPair( makeTimestampFieldname(tContext), getTimeStamp() ), newValues )
        return localUpdateEntry(entryData, newValues)
}
```

Figura E.1: Funciones primitivas: getEntry() y signEntry().

```
// The possible RegisterTypes are AccountLedger, FAM, FBM, Quote, Invoice, Bill
// FieldValues is the type of lists of (fieldName, fieldValue) pairs.
// If the registerType is AccountLedger, FAM or FBM, the value to be used for the
// previousEntry field of the newEntryData must be obtained from the profile before
// calling putEntry.
putEntry(RegisterType rType, FieldValues entryData, String entryIDString, String UUID,
         Boolean sign, TimestampContext tContext, SignatureContext sContext,
         String selfPrivateKey, String otherPublicKey) {
       file entryFile
       if (sign) entryData = signEntry(entryData, tContext, sContext, selfPrivateKey)
       if ( size(entryData) != size(rType) ) { // entryData is incomplete
              entryHash = makeEphemeralDHTHash(UUID, entryData)
              entryFile.write( encrpyt(entryData, otherPublicKey) )
       } else {
              // Pass entryData as parameter in order to use field value transRefHash or
              // FBMEntryNum or FAMEntrynum or ledgerEntryNum, depending on registerType
              entryHash = makePersistentDHTHash(UUID, entryData, rType, entryIDString)
              entryFile.write( entryData )
       }
       localStore(entryIDString, entryHash)
       putDHT(entryFile, entryHash)  // store the content of entryFile in the position entryHash of the DHT
}

// The possible RegisterTypes are AccountLedger, FAM, FBM, Quote, Invoice, Bill
// FieldValues is the type of lists of (fieldName, fieldValue) pairs.
// If the registerType is AccountLedger, FAM or FBM, the value to be used for the
// previousEntry field of the newEntryData must be obtained from the profile before
// calling putEntry.
updateEntry(RegisterType rType, FieldValues newValues, String entryIDString, String UUID,
            Boolean sign, TimestampContext tContext, SignatureContext sContext,
            String selfPrivateKey, String otherPublicKey) {
       previousEntryData = getEntry(entryIDString, UUID)
       entryData = localUpdateEntry(previousEntryData, newValues)
       putEntry(rType, entryData, entryIDString, UUID, sign, tContext, sContext, selfPrivateKey, otherPublicKey)
}
```

Figura E.2: Funciones primitivas: putEntry() y updateEntry().

Bibliografía

[1] *About Timebanks USA*. http://timebanks.org/about/.

[2] E.C. de Almeida y col. "A Framework for Testing Peer-to-Peer Systems". En: *Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on*. 2008, págs. 167-176. DOI: 10.1109/ISSRE.2008.42.

[3] *Amazon S3*. https://aws.amazon.com/es/s3/.

[4] *Amazon.com, Inc*. http://www.amazon.com/.

[5] *Android*. http://es.wikipedia.org/wiki/Android.

[6] Aris M. Ouksel Kai-Uwe Sattler Angela Bonifati Panos K. Chrysanthis. "Distributed Databases and Peer-to-Peer Databases: Past and Present". En: *SIGMOD Record* (2008).

[7] *Apache Cassandra*. http://cassandra.apache.org/.

[8] *Asociación para el Desarrollo de los Bancos de Tiempo*. http://adbdt.org/.

[9] James Aspnes y Gauri Shah. "Skip graphs". En: *ACM Transactions on Algorithms (TALG)* 3.4 (2007), pág. 37.

[10] *Autoridad certificadora*. http://e-administracion.cea.es/autoridades.

[11] *Ayudarnos*. http://www.ayudarnos.org/.

[12] Ozalp Babaoglu, Moreno Marzolla y Michele Tamburini. "Design and Implementation of a P2P Cloud System". En: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. SAC '12. Trento, Italy: ACM, 2012, págs. 412-417. ISBN: 978-1-4503-0857-1. DOI: 10.1145/2245276.2245357. URL: http://doi.acm.org/10.1145/2245276.2245357.

[13] *BDT Online*. http://www.bdtonline.org/.

[14] Hector Garcia-Molina Beverly Yang. "Comparing Hybrid Peer-to-Peer Systems". En: ().

[15] *Bitcoin*. https://bitcoin.org/es/.

[16] *Bitstorm*. https://stormhub.org/tracker/ui.php.

[17] *BitTorrent*. http://www.bittorrent.com/intl/es/.

[18] *Bleep*. http://www.bleep.pm/.

[19] *BTProof*. https://www.btproof.com/.

[20] Edgar Cahn. *Time Banking: An Idea Whose Time Has Come?* http://www.yesmagazine.org/new-economy/time-banking-an-idea-whose-time-has-come.

[21] Charalampos Chelmis. "Mutable Peer-to-Peer File Systems: Analysis and Evaluation". En: *IEEE computer society* (2009).

[22] *CHEs-ROCS System*. http://che-rocs.wikispaces.com/.

[23] *Community Networks Testbed*. https://community-lab.net/.

[24] *Community Weaver*. http://timebanks.org/get-started/community-weaver/.

[25] *Corel Draw X7*. http://www.coreldraw.com/.

[26] Fabrizio Cornelli y col. "Choosing reputable servents in a P2P network". En: *Proceedings of the 11th international conference on World Wide Web*. ACM. 2002, págs. 376-386.

[27] *d-p2p-sim*. https://code.google.com/p/d-p2p-sim/.

[28] Frank Dabek y col. "Wide-area cooperative storage with CFS". En: *ACM SIGOPS Operating Systems Review* 35.5 (2001), págs. 202-215.

[29] Ernesto Damiani y col. "A reputation-based approach for choosing reliable resources in peer-to-peer networks". En: *Proceedings of the 9th ACM conference on Computer and communications security*. ACM. 2002, págs. 207-216.

[30] *Decentralized Trusted Timestamping using the Crypto Currency Bitcoin*. http://www.gipp.com/wp-content/papercite-data/pdf/gipp15a.pdf.

[31] *Design Patterns: Elements of Reusable Object-Oriented Software.* Prentice Hall; Edición: 1st ed., Reprint. (17 de enero de 1980), 1980. ISBN: 978-020-16-3361-0.

[32] *Distributed systems, concepts and designs. 5th edition.* Addison Wesley. Pearson, 2011. ISBN: 978-0-13-214301-1.

[33] Juan Antonio Garrido Domínguez. *Gestión Web para un Banco del Tiempo.* `http://www.lcc.uma.es/~ppgg/PFC/LETS/LETSDoc.pdf`.

[34] John R. Douceur. "The Sybil Attack". En: *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS).* 2002. URL: `http://research.microsoft.com/apps/pubs/default.aspx?id=74220`.

[35] Boris Dragovic y col. "Xenotrust: Event-based distributed trust management". En: *null.* IEEE. 2003, pág. 410.

[36] *draw.io.* `https://www.draw.io/`.

[37] *Dropbox.* `http://www.dropbox.com`.

[38] Peter Druschel y Antony Rowstron. "PAST: A large-scale, persistent peer-to-peer storage utility". En: *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on.* IEEE. 2001, págs. 75-80.

[39] *eBay.* `http://www.ebay.com/`.

[40] *Eclipse IDE.* `https://eclipse.org/ide/`.

[41] *e(fx)clipse.* `http://www.eclipse.org/efxclipse/index.html`.

[42] *Enterprise Architect.* `https://www.sparxsystems.es/`.

[43] *Facebook.* `https://www.facebook.com/`.

[44] *FreePastry.* `http://www.freepastry.org/`.

[45] *FreePastry Simulator.* `https://trac.freepastry.org/wiki/tut_simulator`.

[46] *Gary C. Kessler.* `http://www.garykessler.net/library/crypto.html`.

[47] Madan Jampani Gunavardhan Kakulapati Avinash Lakshman Alex Pilchin Swaminathan Sivasubramanian Peter Vosshall Giuseppe DeCandia Deniz Hastorun y Werner Vogels. "Dynamo: Amazon's Highly Available Key-value Store". En: *Amazon.com* (2007).

[48] P Godfrey y Ion Stoica. "Heterogeneity and load balance in distributed hash tables". En: *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*. Vol. 1. IEEE. 2005, págs. 596-606.

[49] *Google Code*. https://code.google.com/.

[50] *Google Drive*. https://drive.google.com/.

[51] Dr.-Ing. Kalman Graffi. *P2P-based Storage Systems*. http://www.cs.uni-paderborn.de/fileadmin/Informatik/FG-TI/Lehre/WS2011/P2PNets/UPB__WS2011__P2PNetworks__Topic-10__Storage.pdf.

[52] Dr.-Ing. Kalman Graffi. *Structured Heterogeneous P2P Overlay Networks*. http://www.cs.uni-paderborn.de/fileadmin/Informatik/FG-TI/Lehre/WS2011/P2PNets/UPB__WS2011__P2PNetworks__Topic-06__Overlays-Structured-Heterogeneous.pdf.

[53] Dr.-Ing. Kalman Graffi. *Unstructured P2P Overlay Networks*. http://www.cs.uni-paderborn.de/fileadmin/Informatik/FG-TI/Lehre/WS2011/P2PNets/UPB__WS2011__P2PNetworks__Topic-03__Overlays-Unstructured.pdf.

[54] *Hosted RedMine*. https://www.hostedredmine.com/.

[55] *hOurworld*. https://www.hourworld.org/.

[56] *How Ripple Works - Gateways and Pathways - YouTube*. www.youtube.com/watch?v=M16ZatXbmLg.

[57] *Ingeniería del software. Novena edición*. Addison Wesley. Pearson, 2011. ISBN: 978-607-32-0603-7.

[58] *Instituto Nacional de Ciberseguridad*. https://www.incibe.es/blogs/post/Seguridad/BlogSeguridad/Articulo_y_comentarios/trusted_timestamping.

[59] *IntegralCES*. https://www.drupal.org/node/2267557.

[60] *Java*. https://www.java.com/.

[61] *Javadoc*. http://www.oracle.com/technetwork/articles/java/index-jsp-135444.html.

[62] JL Jiménez y col. "Validación de un sistema computacional P2P mediante un estudio empırico". En: ().

[63] *JUnit*. http://junit.org/.

[64] SD Kamvar, MT Schlosser y H Garcia-Molina. "EigenRep: Reputation management in peer-to-peer networks". En: *Proceedings of 12th International World Wide Web Conference, Budapest, Hungary*. 2003.

[65] *LaTeX*. `http://www.latex-project.org/`.

[66] *LatexEditor*. `http://www.latexeditor.org/`.

[67] Seungjoon Lee, Rob Sherwood y Bobby Bhattacharjee. "Cooperative peer groups in NICE". En: *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*. Vol. 2. IEEE. 2003, págs. 1272-1282.

[68] *LinkedIn*. `http://www.linkedin.com/`.

[69] P. Mayer y col. "The Autonomic Cloud: A Vision of Voluntary - Peer-2-Peer Cloud Computing". En: *Self-Adaptation and Self-Organizing Systems Workshops (SASOW) - 2013 IEEE 7th International Conference on*. 2013, págs. 89-94. DOI: `10.1109/SASOW.2013.16`.

[70] Petar Maymounkov y David Mazieres. "Kademlia: A peer-to-peer information system based on the xor metric". En: *Peer-to-Peer Systems*. Springer, 2002, págs. 53-65.

[71] Jill Miller. *Teruko Mizushima: Pioneer Trader in Time as a Currency*. `http://intersections.anu.edu.au/issue17/miller.htm`.

[72] *Modelio*. `https://www.modelio.org/`.

[73] *Monedas complementarias; un itinerario para construir un sistema de intercambio alternativo*. `http://www.bdtonline.org/wiki/images/b/be/Ponencia_Monedas_Complemt_ener2012_LaCabrera.pdf/`.

[74] Tim D Moreton, Ian A Pratt y Timothy L Harris. "Storage, mutability and naming in pasta". En: *Web Engineering and Peer-to-Peer Computing*. Springer, 2002, págs. 215-219.

[75] *NICE*. `http://www.cs.umd.edu/projects/nice/`.

[76] *ns2*. `http://nsnam.isi.edu/nsnam/index.php/Main_Page`.

[77] *Open Chord*. `http://open-chord.sourceforge.net/`.

[78] *OriginStamp*. `http://www.originstamp.org/`.

[79] *OverSim - The Overlay Simulation Framework*. `http://www.oversim.org/`.

[80] *P2P Trust Simulator*. http://rtg.cis.upenn.edu/qtm/p2psim.php3.

[81] *P2PSim*. http://pdos.csail.mit.edu/p2psim/howto.html.

[82] *Papyrus*. https://www.eclipse.org/papyrus/.

[83] *Peer-to-Peer Computing*. Springer-Verlag Berlin Heidelberg, 2010. ISBN: 978-3-642-03513-5.

[84] *Peer-to-Peer Computing. Principles and Applications*. Springer, 2010.

[85] *PeerfactSim.KOM*. https://sites.google.com/site/peerfactsimkom/.

[86] *PeerSim: A Peer-to-Peer Simulator*. http://peersim.sourceforge.net/.

[87] *PeerThing*. http://sourceforge.net/projects/peerthing/?source=navbar.

[88] *PeopleDecide*. https://www.peopledecide.org.au/eserai.

[89] *PlanetLab*. http://www.planet-lab.eu/.

[90] *Principios del Manifiesto Ágil*. http://agilemanifesto.org/iso/es/principles.html.

[91] *Propuesta de banco de tiempo*. https://www.facebook.com/groups/p2ptimebank/.

[92] *Protocolo de pago de Bitcoin*. https://github.com/bitcoin/bips/blob/master/bip-0070.mediawiki.

[93] *Qué es SCRUM*. http://www.proyectosagiles.org/que-es-scrum.

[94] Sylvia Ratnasamy y col. *A scalable content-addressable network*. Vol. 31. 4. ACM, 2001.

[95] *Redmine*. http://www.redmine.org/.

[96] *Ripple*. https://ripple.com/trade/.

[97] Antony Rowstron y Peter Druschel. "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems". En: *Middleware 2001*. Springer. 2001, págs. 329-350.

[98] V.S. Dixit Rupali Bhardwaj Anil Kr. Upadhyay. "An Overview on Tools for Peer to Peer Network". En: ().

[99] *SimGrid*. http://simgrid.gforge.inria.fr/.

[100] *Skype*. https://www.skype.com/.

[101] *Slashdot*. http://slashdot.org/.

[102] IEEE Computer Society. "IEEE Recommended Practice for Software Requirements Specifications". En: *IEEE Std 830-1998* (1998), págs. 1-40. DOI: 10.1109/IEEESTD.1998.88286.

[103] *Source Forge*. http://sourceforge.net/.

[104] *Stack Overflow*. http://stackoverflow.com/.

[105] Ion Stoica y col. "Chord: A scalable peer-to-peer lookup service for internet applications". En: *ACM SIGCOMM Computer Communication Review* 31.4 (2001), págs. 149-160.

[106] *Structured Peer to Peer Systems*. Springer, 2013.

[107] *TeamViewer*. https://www.teamviewer.com.

[108] *Texlipse*. http://marketplace.eclipse.org/content/texlipse/.

[109] *TexStudio*. http://sourceforge.net/projects/texstudio/.

[110] *The Future of Money*. Random House Business; New Ed edition (17 Jan. 2002), 2002. ISBN: 978-0-7126-9991-4.

[111] *The Wörgl Currency and Demurrage*. http://reinventingmoney.com/worgl/.

[112] *Time Overflow*. https://www.timeoverflow.org/.

[113] *Timebanking UK*. http://www.timebanking.org/.

[114] *TimeBanks*. http://timebanks.org/.

[115] *TimeOverflow*. https://www.timeoverflow.org/.

[116] *TimeOverFlow - GitHub*. https://github.com/coopdevs/timeoverflow.

[117] A. Basu S. Rodhetbhai I. Wakeman D. Chalmers TS. Naicken B. Livingston. "The State of Peer-to-Peer Simulators and Simulations". En: ().

[118] Timebanking UK. *About coproduction*. http://www.timebanking.org/what-is-timebanking/about-coproduction/.

[119] Timebanking UK. *What are the benefits of timebanks*. http://www.timebanking.org/what-is-timebanking/what-are-the-benefits-of-timebanks/.

[120] *Visual Paradigm*. http://www.visual-paradigm.com/.

[121] *Voldemort*. http://www.project-voldemort.com/voldemort/.

[122] *WhatsApp*. http://www.whatsapp.com/.

[123] *Wi-Fi Direct*. http://es.wikipedia.org/wiki/Wi-Fi_Direct.

[124] Wikipedia. *Ataque Sybil*. http://en.wikipedia.org/wiki/Sybil_attack.

[125] Wikipedia. *Banco de pruebas - Wikipedia - The Free Encyclopedia*. http://es.wikipedia.org/wiki/Banco_de_pruebas.

[126] Wikipedia. *Chord - Wikipedia - The Free Encyclopedia*. http://es.wikipedia.org/wiki/Chord.

[127] Wikipedia. *Copiador de Unix a Unix - Wikipedia - The Free Encyclopedia*. http://es.wikipedia.org/wiki/Copiador_de_Unix_a_Unix.

[128] Wikipedia. *Diagrama de Gantt*. http://es.wikipedia.org/wiki/Diagrama_de_Gantt.

[129] Wikipedia. *Domain Name System - Wikipedia - The Free Encyclopedia*. http://en.wikipedia.org/wiki/Domain_Name_System.

[130] Wikipedia. *Extreme Programming - Wikipedia - The Free Encyclopedia*. http://en.wikipedia.org/wiki/Extreme_programming.

[131] Wikipedia. *Garbage collection - Wikipedia - The Free Encyclopedia*. http://es.wikipedia.org/wiki/Recolector_de_basura.

[132] Wikipedia. *Gossip protocol*. http://en.wikipedia.org/wiki/Gossip_protocol.

[133] Wikipedia. *Hawala - Wikipedia - The Free Encyclopedia*. http://en.wikipedia.org/wiki/Hawala.

[134] Wikipedia. *JavaFX - Wikipedia - The Free Encyclopedia*. http://es.wikipedia.org/wiki/JavaFX.

[135] Wikipedia. *Kademlia - Wikipedia - The Free Encyclopedia*. http://es.wikipedia.org/wiki/Kademlia.

[136] Wikipedia. *Local exchange trading system - Wikipedia - The Free Encyclopedia*. http://en.wikipedia.org/wiki/Local_exchange_trading_system.

[137] Wikipedia. *Napster - Wikipedia - The Free Encyclopedia*. `http://en.wikipedia.org/wiki/Napster`.

[138] Wikipedia. *Pastry - Wikipedia - The Free Encyclopedia*. `http://en.wikipedia.org/wiki/Pastry_(DHT)`.

[139] Wikipedia. *Peer-to-peer - Wikipedia - The Free Encyclopedia*. `http://es.wikipedia.org/wiki/Peer-to-peer`.

[140] Wikipedia. *Scrum - Wikipedia - The Free Encyclopedia*. `http://es.wikipedia.org/wiki/Scrum`.

[141] Wikipedia. *Time based currency - Wikipedia - The Free Encyclopedia*. `http://en.wikipedia.org/wiki/Time-based_currency`.

[142] Wikipedia. *Usenet - Wikipedia - The Free Encyclopedia*. `http://es.wikipedia.org/wiki/Usenet`.

[143] *WindowBuilder*. `http://www.eclipse.org/windowbuilder/`.

[144] *WordPress Timebank System*. `http://www.time-bank.info/`.

[145] Li Xiong y Ling Liu. "Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities". En: *Knowledge and Data Engineering, IEEE Transactions on* 16.7 (2004), págs. 843-857.

[146] Ben Yanbin Zhao, John Kubiatowicz, Anthony D Joseph y col. "Tapestry: An infrastructure for fault-tolerant wide-area location and routing". En: (2001).

[147] Philip R Zimmermann y Philip R Zimmermann. *The official PGP user's guide*. Vol. 265. MIT press Cambridge, 1995.